AN SLA REALIZATION OF THE 6502 MICROPROCESSOR

by

Kenju Tsuyuki

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

1981

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SINGED: Tenji Tsuyuki

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Fredrick J. Hill
Fredrick J. Hill
Professor of
Electrical Engineering

14 Dec. 1981
Date

ACKNOWLEDGMENTS

I would like to express my deep appreciation to Professor Fredrick J. Hill for his guidance and advice during the preparation of this thesis.

I also wish to thank all those who participated in the hardware design language research project for their cooperation.

TABLE OF CONTENTS

TABLE OF CONTENTS -- Continued

TABLE OF CONTENTS -- Continued

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

The fabrication of a complete microprocessor on a single VLSI chip is now standard practice. Attempting to achieve this goal using manual design methods is not a practical proposition. Computer aids play an important role in the development and design of such single-chip microprocessors.

This research project investigates the use of a hardware design language, AHPL, as the front end to a design automation process for a microprocessor. The popular MOS Technology 6502 is chosen as the object of the investigation.

After a detailed analysis of the internal operations and instruction set of the 6502, an AHPL description of the 6502 is formulated. This is followed by a simulation using HPSIM, the AHPL simulator, to debug and refine the design.

As a final exercise, a scheme is proposed for the translation of the AHPL description into a Storage Logic Array form of realization.

CHAPTER 1

INTRODUCTION

In recently published books and magazines on computers and digital systems, the term, design automation can often be encountered. The design automation as well as the term, computer-assisted design generally implies design method or a tool that can assist designers in designing or developing products in various fields such as digital hardware systems, computers of all sizes, and integrated circuits in the field of digital system design. The design automation can be applied to almost all areas of design processes because it is desirable to shorten the turnaround time of design.

Especially, for the design of very large scale integration (VLSI) used in digital systems, the use of the regularity of patterns for communication between active elements is absolutely essential to implement a design automation system. Although resultant products realized by the design automation system tend to have reduntant regular patterns, this redundancy can be justified by the feasibility of the design automation system, the shorter turnaround time of design, and the simplicity of design

process.

For the purpose of the design of digital hardware systems, hardware design languages have been invented. The use of a hardware design language is one of the important elements of a design automation system. A Hardware Programming Language (AHPL), extensively described in reference 1 has been widely in use to design digital systems. Some compiler programs have been implemented in order to aid the design of the digital systems described in AHPL since AHPL was introduced. However, a new multi-stage compiler has been written in connection with several new notations and concepts added in the existing AHPL. This universal AHPL will be a useful tool for not only the design of conventional digital systems but also a VLSI implementation.

Another important element of a design automation system designed for a VLSI implementation is to use a regular pattern for a basic element. A Storage Logic Array (SLA), adequately described in reference 2 has been examined for the use of a VLSI implementation, and the study of a compiler capable of translating from an AHPL description to a VLSI implementation in the form of the SLA has also been carried out. As a result, it has been proved that the compiler is feasible and the use of clocked D flip-flops as the storage elements in the SLA is more natural.

In the SLA compilation project, an algorithm (stage 3) to translate the output of stage 2 of the new compiler into the data used for an SLA realization must be developed. In addition, some design examples should be done so that the results of the examples can be fed back to the design of the algorithm. For this reason, the 6502 microprocessor, as an example of an SLA realization, was selected.

Since the advent of the first microprocessor in 1971, many microprocessors have been developed. In spite of the complicated operations of the microprocessors, they are usually designed by menas of a random logic technique. The 6502, an 8-bit microprocessor, is a sophisticated digital system itself and has many powerful instructions. Therefore, the design example of the 6502 will be able to satisfy most of the requirements on establishing a design automation system aimed at the SLA realization. The 6502 is also sufficiently complicated to confirm the completeness of the design automation system.

In implementing the 6502 microprocessor in the form of the SLA, the following tasks must be worked out:

1) Detailed Analysis of the internal operations of the 6502.
2) The systematic development of a 6502 AHPL description.

3) Simulation of the 6502 written in AHPL.

4) Translation from the 6502 AHPL description to an SLA form by either manual work or a computer.

Through this paper, the first three tasks are discussed with emphasis. An introduction to the SLA and the stage 3 algorithm is briefly mentioned in conjunction with the task of the translation.

Chapter 2 presents a brief description of the hardware organization of the 6502 to prepare the background for the discussion in later chapters. Chapter 3 analyzes the internal operations of the 6502, and a 6502 AHPL description is developed step by step in the following chapter. In Chapter 5, the preparation for simulation and the results of the simulation are explained. Chapter 6 mentions an introduction to an SLA along with an algorithm for an SLA realization briefly. The results of a design example are discussed in the last chapter.

Finally, Appendix A provides a 6502 instruction set summary, and Appendix B presents a summary of cycle by cycle operation of instructions. Appendix C lists internal operations of typical instructions, while Appendix D describes internal operations of inputs. A Karnaugh map of operation codes is included in Appendix E. Appendix F shows a complete 6502 AHPL description, while Appendix G lists combinational logic unit descriptions. The last appendix,

Appendix H, demonstrates an example of simulation.

CHAPTER 2

THE 6502 MICROPROCESSOR

The purpose of this chapter is to give the reader an overview of the 6502 so that the reader can proceed to the following chapters with an equal knowledge of the 6502. In order to present a clear description, this chapter is divided into two sections, Hardware Organization and Instruction Set for descriptions of a hardware aspect and a software aspect, respectively.

Since the first microprocessor appeared on the market of digital hardware systems, many have been introduced. The 6502 that is one of the third generation microprocessors was introduced after the 6800 microprocessor had been put on the market. Because of the time of appearance, the 6502 has many similar features to the 6800.

Hardware Organization

The hardware organization and the instruction set of the 6502 are very similar to those of the 6800. The is due to the fact that the design of the 6502 was based on the 6800. Like many 8-bit microprocessors, the 6502 has also an 8-bit data bus and a 16-bit address bus. This indicates the capability of addressing 64 Kbytes (8 bits) of memory. In

addition to the data bus and address bus, there are nine inputs and four outputs used in the 6502. All inputs and outputs are put together into a 40-pin dual-in-line package. There are three unused pins on the packege. The pin configuration is shown in Figure 1.

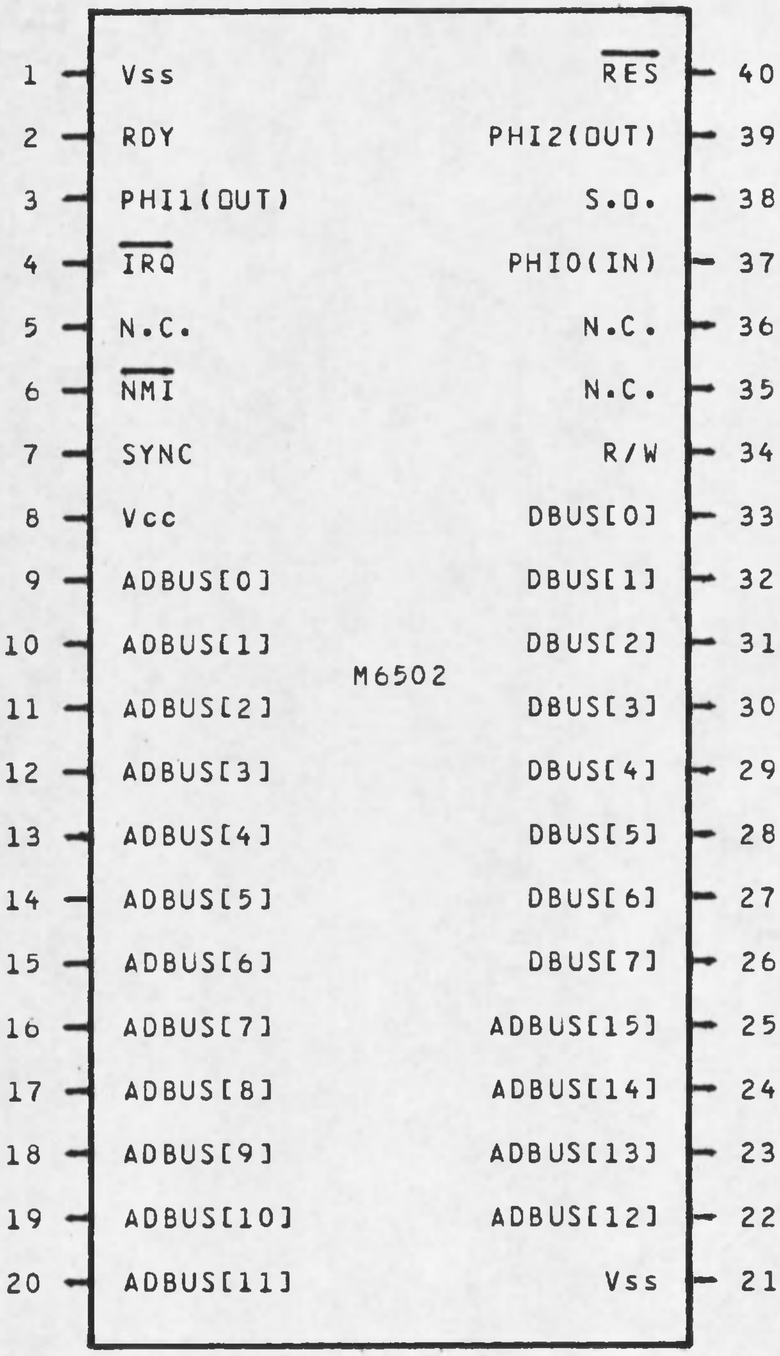
Inputs and Outputs

The inputs and outputs consist of an 8-bit data bus, a 16-bit address bus, $\overline{IRQ}$, $\overline{NMI}$, $\overline{RES}$, RDY, PHIO, S.O., R/W, SYNC, PHI1, PHI2, two Vss's, and Vcc. The inputs and outputs have their own distinct operations. Typical features and operations of the inputs and outputs will be briefly explained as follows:

ADBUS[0:15] -- The 16-bit output bus is used to address locations allocated in memory and peripherals and is compatible with TTL.

DBUS[0:7] -- The 8-bit bi-directional bus with the tri-state output is provided as a medium of transferring data. For example, the microprocessor puts data onto the data bus for write operations and gets data from the bus for read operations.

$\overline{IRQ}$ -- The $\overline{IRQ}$ input is one of the interrupt inputs provided in the 6502. The TTL level input requests an interrupt sequence in which the Program Counter (PC) and the Processor Status Register (P) will be stored in the stack. At the end of the sequence, the data from the location FFFE



N.C. = No Connection.

Figure 1. Pin Configuration of the 6502

and the data from the location FFFF will be loaded into the program counter low and the program counter high, respectively for the starting location of an interrupt routine. This is a level sensitive input.

$\overline{\text{NMI}}$ -- The $\overline{\text{NMI}}$ input is also used for an interrupt. The only difference from $\overline{\text{IRQ}}$ is the use of the vector locations FFFA and FFFB, a non-maskable interrupt sequence, and an edge sensitive input.

$\overline{\text{RES}}$ -- The $\overline{\text{RES}}$ input is used to reset or start the 6502 from a power down condition. As in the interrupt sequence, the starting location for program control will be loaded from the vector locations FFFC and FFFD.

RDY -- By the RDY input, all cycles except write cycles can be halted. This feature allows the microprocessor to be connected with a low speed memory.

PHI0 -- This PHI0 input is used to generate a two phase non-overlapping clock. An internal clock generator is supplied inside the 6502.

S.O. -- The overflow bit in the processor status register can be set by a negative going edge on the Set Overflow Flag (S.O.) input.

R/W -- The R/W output is used to distinguish between those cycles in which the microprocessor is doing a read operation and those cycles in which the processor is doing a write operation. This output can also be used to control

read and write operations of memory modules and peripheral devices.

SYNC -- The SYNC output is provided to identify those cycles in which the microprocessor is doing an OP CODE fetch.

PHI1 & PHI2 -- The phase 1 clock and phase 2 clock of the two phase non-overlapping clock generated in the 6502 are output on PHI1 and PHI2, respectively to be used by other devices.

By now, all of the inputs and outputs have been briefly described except for Vss and Vcc that are used for power supply. The inputs and outputs are summarized in Table 1 for clarity. Detailed information concerning the internal operations of $\overline{\text{IRQ}}$, $\overline{\text{NMI}}$, and $\overline{\text{RES}}$ and the exact timing of the inputs and outputs should be referred to in reference 3 for further understanding.

Internal Buses

The data bus and address bus are provided in the 6502 so that the microprocessor can communicate with outside devices. On the other hand, internal buses are employed in order that data can move inside the microprocessor. There are three internal buses, namely the Internal Address Bus (IADBUS), the Internal Data Bus (IDBUS), and the Argument Bus (ARGBUS). These buses will be simply explained in connection with the hardware organization.

Table 1. Summary of Inputs and Outputs

Signal Name	Input/Output	Features
ADBUS[0:15]	Output	TTL Compatible.
DBUS[0:7]	Input/Output	Bidirectional Bus with Tri-state Output.
$\overline{\text{IRQ}}$	Input	Level Sensitive.
$\overline{\text{NMI}}$	Input	Edge Sensitive.
$\overline{\text{RES}}$	Input	Level Sensitive.
RDY	Input	TTL Compatible.
PHI0	Input	TTL Compatible.
S.O.	Input	TTL Compatible.
R/W	Output	TTL Compatible.
SYNC	Output	TTL Compatible.
PHI1	Output	TTL Compatible.
PHI2	Output	TTL Compatible.

IADBUS[0:15] -- The internal address bus consists of two 8-bit buses, IADBUS[0:7] and IADBUS[8:15]. The bus is connected to the ADBUS through the address buffers so that internally generated addresses can be sent to the ADBUS. Furthermore, the bus is used to route data to the Increment/ Decrement Register (INR) or the Incrementer/Decrementer (INC/DEC). The data will be used in later operations. The outgoing data on the IADBUS will come from internal registers such as the Data Latch (DL), the Temporary Register (TR), INR, PC, the ALU Register (ALUREG), and the Stack Pointer (SP) to make up 16-bit addresses put onto the ADBUS. Only the IADBUS[8:15] can be connected to the ARGBUS, so the data on the IADBUS[8:15] can be an argument of the ALU for arithmetic operations.

IDBUS[0:7] -- Unlike the IADBUS, the internal data bus is connected to almost all internal registers. The IDBUS is mainly used as a medium of transferring data between the registers. The data on the IDBUS can be loaded onto the DBUS through the data bus buffer. The data coming from outside must be latched into the data latch first.

ARGBUS[0:7] -- The 8-bit argument bus is prepared for routing the data from one of the sources, the Index X Register (X), the Index Y Register (Y), and the IADBUS[8:15] to the ALU. The ARGBUS is one of the four arguments of the ALU.

These buses are closely connected to the internal registers and combinational logic units (CLU's), which will be explained next. A pictorial representaion of the internal buses can be seen in Figure 2.

Internal Registers

The internal registers as well as the internal buses are quite significant to the hardware organization. Eleven registers have been utilized to carry out the internal operations of all instructions and input signals.

As already noticed, the pictorial view of 6502 internal architecture is shown in Figure 2. However, this figure is not completely the same as those available in manuals and books on the 6502. This is because those pictures do not reflect real design and some modifications had to be done for the 6502 AHPL description to be discussed in this paper. Even among those pictures, there are still some differences. A major difference between Figure 2 and the others is that the temporary register is depicted in Figure 2. Another is the explicitly indicated ARGBUS. Individual registers will be explained in brief as follows:

DL -- The data latch is used to latch incoming data on the DBUS for data fetches. The DL will also load the data onto either the IDBUS for transfers or the DBUS for write operations. The data fetches will occur depending upon the instruction being performed by the processor. The

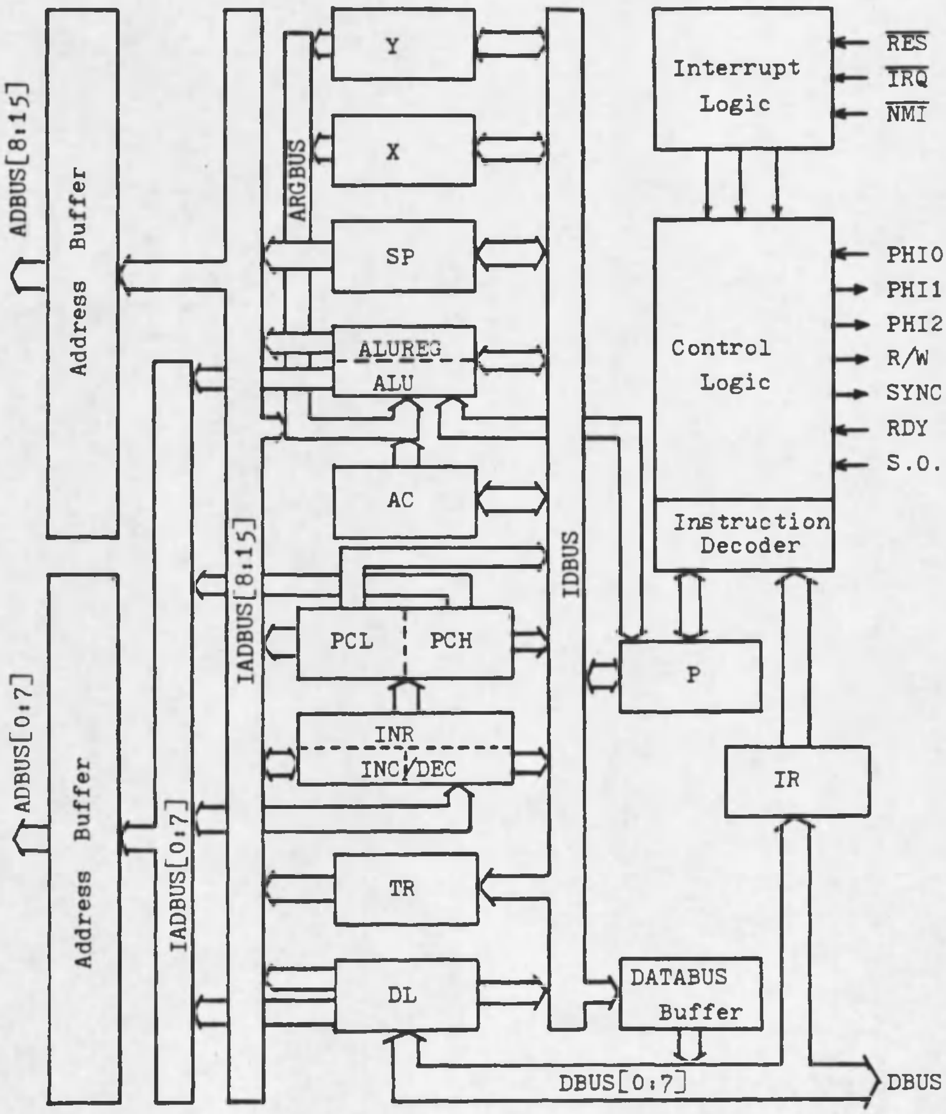


Figure 2. 6502 Internal Architecture

data stored in DL will be used either as half portions of addresses or in later operations.

TR -- The temporary register is provided to save data that will be used as half parts of addresses in later operations.

IR -- The data fetched from outside devices in the OP CODE fetch cycle will be stored in the instruction register. The data will in turn be sent to the instruction decoder to generate appropriate control signals.

P -- The processor status register is provided to indicate the current status of the processor. The P register is further partitioned into the Carry (C), Zero (Z), Interrupt Disable (I), Decimal Mode (D), Break Command (B), Overflow (O), and Negative (N) flags to be used by branch instructions, arithmetic and logic operations, etc.

INR -- The increment/decrement register stores data from either INC/DEC or IADBUS. Data stored in INR will usually be used for addressing.

PC -- The program counter is made up of the program counter low and program counter high. Like INR, Data stored in PC will be used for addressing.

AC -- The accumulator, as the name implies, accumulates most of the computed data from the ALU. The contents of the accumulator can also be an argument of the ALU for arithmetic and logic operations.

ALUREG -- The ALU register is a temporary storage for the data coming out of the ALU. The ALUREG will be refreshed every cycle. Therefore, ALUREG can not store data for more than one cycle. The contents of the register can be half parts of addresses put onto the IADBUS.

SP -- The stack pointer is used to point to and save the location currently used in the stack. In order to set the stack pointer, the data on the IDBUS can be stored in SP.

X -- The index X register is prepared for the addressing modes associated with the X register. This register can be used as a temporary storage from programmers' viewpoint. The data in the X register can be loaded onto either the IDBUS or ARGBUS.

Y -- The index Y register is prepared for the addressing modes associated with the Y register. This register can be used as a temporary storage as well. The data in the Y register can be loaded onto either the IDBUS or ARGBUS.

The buses and registers provided inside the 6502 have been described, yet a description of the combinational logic units, the ALU and INC/DEC is not presented. The INC/DEC will not need an explanation at all because it is a mere incrementer/decrementer unit that will increment or decrement address data on the IADBUS. However, the ALU will

require a comprehensive specification. In Chapter 3, the ALU will be discussed in conjunction with arithmetic and logic operations.

Detailed information about the input and output connections of the registers and CLU's is provided in Table 2. Table 2 will compensate for the limitation of the illustration of Figure 2. Thus, exact connections should be looked into in Table 2. In fact, Table 2 has been updated as the AHPL description for the 6502 has been modified. In the next chapter, the implication of these connections will be clarified, and Figure 2 and Table 2 will often be referred to for further discussion.

Instruction Set

The 6502 microprocessor has a powerful instruction set, which is very similar to the 6800 microprocessor's instruction set. Because of the completeness and similarity, the user can write efficient programs and translate a program for the 6800 into a program for the 6502 easily. The 6502 instruction set consists of 151 operation codes (OP CODE's), and these OP CODE's can be classified into two categories, instruction and addressing mode. 56 kinds of instructions and 11 addressing modes are being used. A brief description of the instructions and addressing modes will be given below for the discussion in Chapter 3.

Table 2. Input and Output Connectins of Registers and CLU's

Register or CLU	Input/Output	Connections and Transfers
Input Data Latch (DL)	Input	DL <= DBUS
	Output	IDBUS = DL IADBUS[0:7] = DL IADBUS[8:15] = DL
Instruction Register (IR)	Input	IR <= DBUS
	Output	Instruction Decoder = IR
Temporary Register (TR)	Input	TR <= IDBUS
	Output	IADBUS[8:15] = TR
Processor Status Register (P)	Input	P <= IDBUS P <= CLU Output of the ALU Output P <= Instruction Decoder
	Output	PS = P[4],P[7] IDBUS = P
Stack Pointer (SP)	Input	SP <= IDBUS
	Output	IDBUS = SP IADBUS[8:15] = SP
Index Register X (X)	Input	X <= IDBUS
	Output	IDBUS = X ARGBUS = X
Index Register Y (Y)	Input	Y <= IDBUS
	Output	IDBUS = Y ARGBUS = Y

Table 2. -- Continued

Register or CLU	Input/Output	Connections and Transfers
Arithmetic Logic Unit (ALU)	Input (Argment)	ALU : IDBUS ALU : ARGBUS ALU : OP ALU : PS
	Output	ALUREG <= ALU[1:8] P <= CLU Output of the ALU Output.
Accumulator (AC)	Input	AC <= IDBUS
	Output	IDBUS = AC ARGBUS = AC
ALU Register (ALUREG)	Input	ALUREG <= ALU[1:8]
	Output	IDBUS = ALUREG IADBUS[0:7] = ALUREG IADBUS[8:15] = ALUREG
Program Counter (PC)	Input	PC <= INR
	Output	IDBUS = PC[0:7] IDBUS = PC[8:15] IADBUS = PC
Increment/ Decrement Register (INR)	Input	INR[0:7] <= IADBUS[0:7] INR[8:15] <= IADBUS[8:15] INR[8:15] <= INC[8:15](IADBUS) INR[8:15] <= DEC[8:15](IADBUS) INR <= INC(IADBUS)
	Output	PC <= INR IADBUS[0:7] = INR[0:7] IADBUS[8:15] = INR[8:15] IDBUS = INR[0:7]

Table 2. -- Continued

Register or CLU	Input/Output	Connections and Transfers
Incrementer/ Decrementer (INC/DEC)	Input (Argment)	INC : IADBUS DEC : IADBUS
	Output	INR[8:15] <= INC[8:15](IADBUS) INR[8:15] <= DEC[8:15](IADBUS) INR <= INC(IADBUS)

Fundamental Instructions

For a more concise description, the 56 instructions can be further divided into nine fundamental instructions. They are arithmetic, logic, load, store, branch, jump, set/clear, transfer, and stack instructions. The instructions will be described in terms of the fundamental instructions.

Arithmetic Instruction -- The arithmetic instruction uses the ALU, so that the data stored in AC and DL will be either added or subtracted. The capabilities of addition and subtraction are provided in the ALU for such instructions as ADC, CMP, CPX, CPY, DEC, DEX, DEY, INC, INX, INY, and SBC. Some flags of the P register will be refreshed as a result of an arithmetic operation.

Logic Instruction -- The logic instruction also uses the ALU to perform a logic operation on the data in AC and DL. The functions of AND, OR, exclusive OR, shift, and rotation are furnished in the ALU as well. The logic instruction also affects the P register. The instructions classified into the logic instruction are AND, ASL, BIT, EOR, LSR, ORA, ROL, and ROR. Note that the shift and rotation functions are classified into this instruction for the simple discussion in Chapter 3, although the functions are not logic functions.

Load Instruction -- The load instruction implies

that data trapped by DL will be stored in the X register, Y register, or accumulator. In addition, the zero and negative flags of the P register may be changed depending on the contents of the data. The instructions placed under the load instruction are LDA, LDX, and LDY.

Store Instruction -- The processor will execute the store instruction by loading data stored in the X register, Y register, or accumulator onto the DBUS. The STA, STX, and STY instructions are categorized into the store instruction.

Branch Instruction -- The processor will branch to the location where the next OP CODE fetch will be executed if a branch condition is satisfied. The branch instruction makes use of the carry, zero, overflow, and negative flags for a branch condition. The branch location is calculated in the ALU. The instructions classified under the branch instruction are BCC, BCS, BEQ, BMI, BNE, BPL, BVC, and BVS.

Jump Instruction -- By the jump instruction, the processor will jump unconditionally to the location where the next OP CODE fetch will be done. Unlike the branch instruction, this instruction includes rather complicated operations for writing data in the stack and retrieving data from the stack. The instructions put under the jump instruction are BRK, JMP, JSR, RTI, and RTS. Only the JMP instruction does not require the use of the stack.

Set/Clear Instruction -- The set/clear instruction

is furnished to modify the P register. By this instruction, the carry, decimal mode, and interrupt disable flags can be set or cleared, while the overflow flag can only be cleared. The overflow flag will be set in case that an overflow condition occurs as a result of an arithmetic operation. The other flags are not accessible to the programmer. The instructions classified into the set/clear instruction are CLC, CLD, CLI, CLV, SEC, SED, and SEI.

Transfer Instruction -- The transfer instruction is provided to transfer data from one of the four registers, the X register, Y register, stack pointer, and accumulator into another register of these. Only six combinations of source and destination registers are implemented. The instructions placed under the transfer instruction are TAX, TAY, TSX, TXA, TXS, and TYA. All the instructions except TXS affect the zero and negative flags.

Stack Instruction -- The stack instruction uses the stack pointer in order to put data in the stack or retrieve data from the stack. This instruction is similar to the load and store instructions but uses the stack pointer instead of the program counter for addressing. However, the stack instruction utilizes PC once the currently used stack address has been set in PC. The instructions classified under the stack instruction are PHA, PHP, PLA, and PLP.

The reader should refer to reference 4 for further

information because the above description is not intended to present a complete explanation of the instruction set. Although each fundamental instruction does not describe exactly the execution part of the internal operation for each instruction, it will help to understand details of the instruction set. A 6502 instruction set summary has been provided in Appendix A for readers' convenience and will itself explain the execution part of each instruction more accurately. Typical internal registers are illustrated in Figure 3 to indicate the bit cofiguration adopted in the 6502 AHPL description.

Addressing Modes

As stated before, the 6502 was introduced after the 6800. As a result of the late appearance, versatile addressing modes and efficient instruction cycles were implemented. Programs such as a program that manipulates tables and arrayscan be written very effectively by means of these addressing modes. Moreover, the microprocessor can execute instructions faster because of shorter execution time. Since a full description of the addressing modes has been left to the next chapter, each addressing mode will be simply described in accordance with the purpose of this chapter.

Immediate -- The instructions with the immediate addressing mode contain two bytes (8 bits), an OP CODE and a

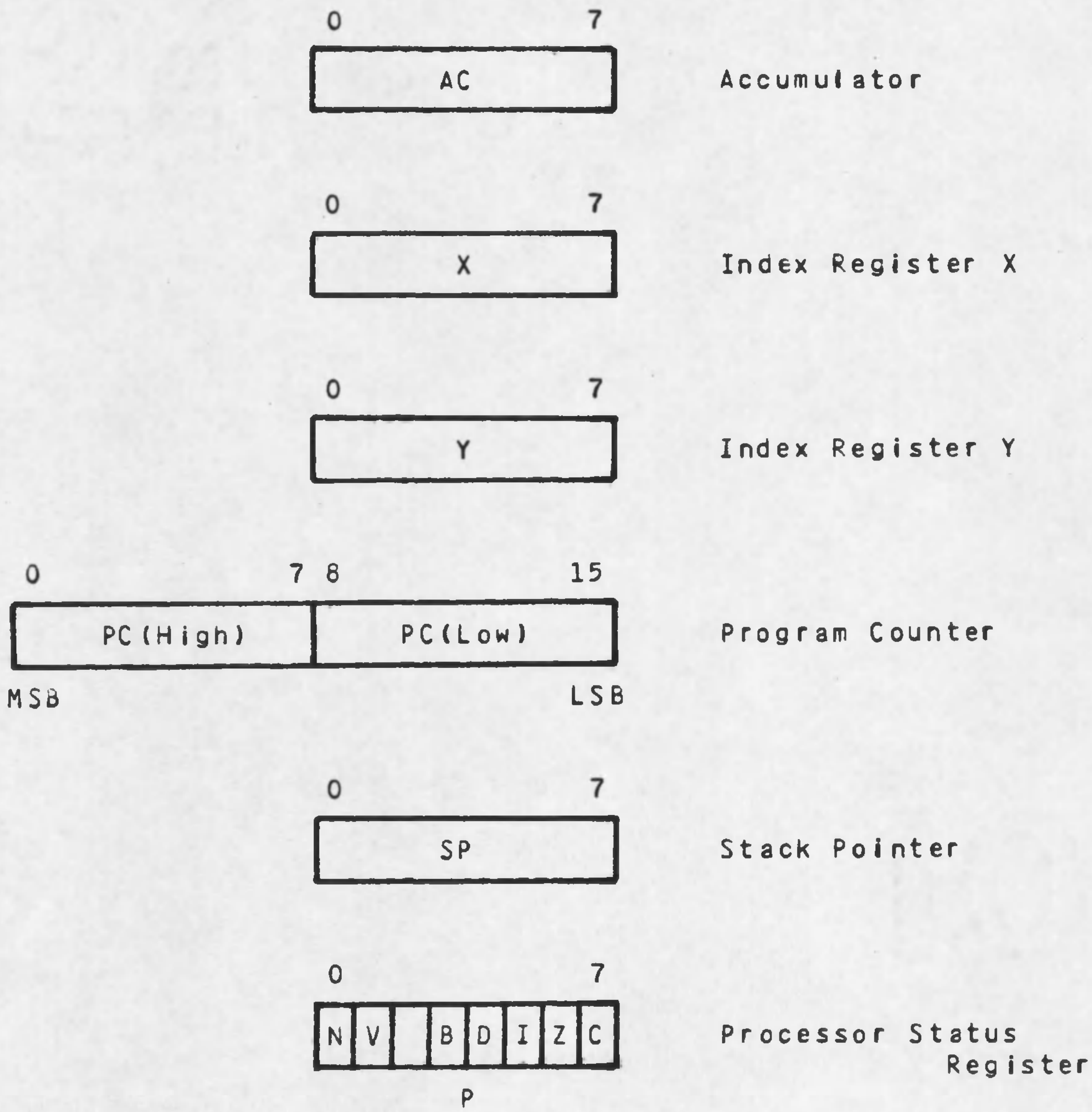


Figure 3. Bit Configuration of Typical Registers

constant value. The OP CODE specifies the internal operation and addressing mode, and the constant value is used in a program. This mode allows the programmer to specify values necessary for the program. The OP CODE and value are fetched from consecutive locations of program memory.

Absolute -- The instructions using the absolute addressing mode use three bytes. The first byte contains an OP CODE for specifying the internal operation and addressing mode. The second byte includes the low order byte of an effective address, while the third byte includes the high order byte of the effective address. Thus, a full 16-bit address can be specified by this mode. The effective address, a full 16-bit address, will be used to address a location at which data will be retrieved and stored.

Zero Page -- The instructions with the zero page addressing mode are made up of two bytes. The first byte is an OP CODE as before, the second byte contains an effective address in page zero of memory. The microprocessor generates all 0's used for the high order byte of a full 16-bit address. This mode will allow the programmer to shorten both memory space and execution time. Only locations in page zero can be used for read and write opeations.

Accumulator -- The instructions having the

accumulator addressing mode include only an OP CODE that indicates the internal operation and addressing mode. This mode can be considered to be the same as the implied mode described below because both the instructions contain only an OP CODE. However, the accumulator addressing mode is categorized distinctly because only the accumulator of the registers accessible to the programmer is utilized for the internal operations of the instructions using this mode.

Implied -- As mentioned above, the instructions with the implied addressing mode use only an OP CODE for specifying the internal operation and addressing mode. Some instructions using the mode require more than 2 cycles. Explanations of these instructions are beyond the scope of this chapter.

Indexed Indirect -- The instructions using the indexed indirect addressing mode contain an OP CODE and a base address in page zero. First, the base address is added to the data stored in the X register. The resultant byte becomes another address in page zero that contains the low order byte of an effective address, while its consecutive address contains the high order byte of the effective address. The effective address will be used for read and write operations.

Indirect Indexed -- The instructions with the indirect indexed addressing mode contain an OP CODE and an

indirect address in page zero. The indirect address contains a base address low, and the next consecutive address of the indirect address contains a base address high. While the base address high is being fetched, the base address low is added to the data stored in the Y register. The resultant byte becomes the low order byte of an effective address. If a carry is generated from the addition, the carry, as binary one, is added to the base address high to increment the address by 1. The resultant byte becomes the high order byte of the effective address. The base address high will be used as the high order byte if no carry.

Zero Page Indexed -- The instructions with the zero page indexed addressing mode include an OP CODE and a base address low. While data is being fetched from the location specified by the base address low in page zero, the base address low is added to the data stored in either the X register or Y register in order to get an effective address in page zero. The data fetched is not used in later operations. No page crossing is allowed in this mode.

Indirect -- The instructions with the indirect addressing mode contain three bytes like those with the absolute addressing mode. The second byte contains the low order byte of an indirect address, while the third byte contains the high order byte of the indirect address. Once

the indirect address has been set up, the low order byte and high order byte of an effective address are fetched from the locations specified by the indirect address and its consecutvie address, respectively. The effective address will be used for the next OP CODE fetch.

Absolute Indexed -- The instructions using the absolute indexed addressing mode include three bytes, an OP CODE, a base address low, and a base address high. The facility to add the base address low to either the X register or Y register is provided to compose the low order byte of an effective address. To carry out a page crossing, the technique to add a carry to the base address high is used.

Relative -- The instructions with the relative addressing mode consist of an OP CODE and an offset. An effective address is calculated by means of the address of the location where the next OP CODE fetch will be performed if a branch is not taken and the offset. The page crossing can occur in this mode.

Further information regarding the addressing modes is extensively described in reference 4.

CHAPTER 3

INTERNAL OPERATIONS ANALYSIS

In the preceding chapter, a brief description of the 6502 microprocessor has been presented. While the description will give the reader a basic knowledge of the 6502, it will not be enough for the development of a 6502 AHPL description. Sufficient information concerning the external behavior of the 6502 must be obtained so that the internal operations of the 6502 can be analyzed for the development. The information can be found in references 3 and 4. A summary of cycle by cycle operation of the 6502 has been produced from the information available in the above references and has been attached in Appendix B. Appendix B will be helpful in analyzing the internal operations of all instructions.

The purpose of this chapter is to analyze the internal operations of all instructions and inputs in order that a 6502 AHPL description can be efficiently and systematically developed. However, it will not be an easy task to analyze the internal operations of all instructions individually because the instruction set has as many as 151 OP CODE's. Since the internal operation of each instruction

consists of an addressing part and an execution part, the internal operations of all instructions can be compactly analyzed in terms of the addressing parts and the execution parts of the internal operations.

Hypotheses

Before proceeding with the analysis of the internal operations, several hypotheses will be defined in order to make the 6502 AHPL description closer to the actual hardware organization of the 6502. These hypotheses must be taken into account in analyzing the internal operations, because the AHPL description will be based directly on the results of this analysis. The hypotheses will be defined as follows:

The Connections to ALU -- The ARGBUS must always be connected to argument 1 of the ALU, while the IDBUS must be connected to argument 2. This is because of avoiding the necessity of additional internal buses. The other two arguments are connected to the P register and instruction decoder. The connection between the ARGBUS and argument 2 and the connection between the IDBUS and argument 1 are not permissible.

ALUREG -- The ALUREG can not store data for more thanone cycle. This means that ALUREG can not be used as a temporary register unless the data will be stored in ALUREG again during the next cycle. Only the data from the ALU can

be stored in ALUREG.

The Use of TR -- Owing to the hypothesis of the connections to the ALU, the use of the temporary register is assumed to save data used for addressing in later operations.

IADBUS[8:15] to ALU -- The data on the IADBUS[8:15] is allowed to be transmitted to the ALU over the ARGBUS for an arithmetic operation. The connection between the IADBUS[0:7] and ARGBUS is not allowed.

These hypotheses are also defined in Figure 2 and Table 2. Since the other connections and operational features not included in the hypotheses can be easily derived from the information available in references 3, 4, and 5, these connections and operational features were defined in advance without question. By adding the hypotheses, the connections between the internal buses, internal registers, and CLU's have been completely defined. Now, the internal operations can be analyzed by means of the completely defined internal architecture.

General Sequence

As all instructions have common operation cycles such as the OP CODE fetch cycle and the following data fetch cycle, it is possible to build a general sequence of the internal operations of all instructions. An internal operation sequence of each internal operation can be written

by using the general sequence as a basic form. The general sequence is given in Figure 4. To prepare for the later discussion on the addressing parts and execution parts, each cycle of the general sequence will be explained as follows:

T0 -- The T0 cycle is the OP CODE fetch cycle and is common to the internal operations of all instructions. To fetch an OP CODE from program memory, the following operations must occur: First, the processor puts a 16-bit address on the IADBUS, and places logic 1 on the SYNC and R/W output lines to indicate the execution of the OP CODE fetch cycle and a read operation. Then, the OP CODE on the DBUS is trapped into IR. During the cycle, the address on the IADBUS is transferred into INC. The address incremented in INC is stored in INR for later operations. Internal registers that are to provide an address on the IADBUS will be determined by the preceding instruction. The PC is selected in the T0 cycle without any meanings.

The above opeations are concisely described in AHPL in Figure 4. In internal operation sequences, owing to the naming convention of the HPSIM2 simulator [6], closely related symbol names are adopted for the actual input and output. For instance, the R/W outputs is named RW here. In addition, the IADBUS is implicitly connected to the ADBUS. The above consideration is because the internal operation sequences derived from the general sequence can be

```
T0:          IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.

            +--------------------------------------------+
Tn-1:       |                                            |
            |      Operations for                        |
Tn:         |      the Addressing and Execution Parts    |
            |                                            |
Tn+1:       |                                            |
            +--------------------------------------------+

T0[OVLP]:    IADBUS =  +------------------+
                       | PC;              |
                       | INR;             |
                       | DL,TR;           |
                       | INR[0:7],ALUREG; |
                       | ALUREG,INR[8:15];|
                       +------------------+
             IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).
            +--------------------------------------------+
            |   Operations for the Execution Part        |
            +--------------------------------------------+

T1[OVLP]:    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.
            +--------------------------------------------+
            |   Operations for the Execution Part        |
            +--------------------------------------------+
```

Figure 4. General Sequence for Instructions

directly utilized for the development of the 6502 AHPL description.

T1 -- The T1 cycle is the data fetch cycle following the OP CODE fetch and is also common to the internal operations of all instructions. First, the address stored in INR is placed on the IADBUS for addressing. At the same time, the processor puts logic 0 on the SYNC line and logic 1 on the R/W line to indicate the execution of a non-OP CODE fetch cycle and a read operation. Next, as in the T0 cycle, the address on the IADBUS is incremented in INC and then stored in INR. Before deleted by the new address from INC, the address previously stored in INR must be transferred into PC to be used in later operations.

Tn-1, Tn, and Tn+1 -- These cycles start at the T2 cycle and end immediately before the T0[OVLP] cycle. The number of the cycles varies depending upon each instruction. For example, the instructions using the immediate addressing mode do not need such cycles, while the instructions employing the indirect indexed addressing mode require at least three cycles. These cycles are provided to specify operations for an addressing part and execution part. Some instructions such as the JMP instruction and branch instructions do not require any operations for the execution parts of their internal operations.

T0[OVLP] -- The T0[OVLP] cycle is the OP CODE

fetch cycle of the next instruction and is exactly the same as the T0 cycle. However, to distinguish the OP CODE fetch cycle of the next instruction from the OP CODE fetch cycle of the current instruction, the space for specifying operations for the addressing part and execution part defined by the current instruction has been provided in the T0[OVLP] cycle. With respect to the addressing part in the T0[OVLP] cycle, registers to supply a 16-bit full address to the IADBUS are determined by the current instruction. Some operations for the execution part can overlap the T0[OVLP] cycle if necessary. However, the conflict on the internal buses, internal registers, and CLU's must be avoided. The IDBUS is usually available for the execution part during this cycle.

T1[OVLP] -- The T1[OVLP] cycle has the same operations as the T1 cycle has. The T1[OVLP] cycle can also be overlapped by the execution part of the preceding instruction, as in the T0[OVLP] cycle.

Since it is assumed that the OP CODE fetched during the T0 cycle is the first instruction and no instruction precedes that instruction, the T0 and T1 cycles could not be overlapped by an execution part. This is the reason that only the T0[OVLP] and T1[OVLP] cycles can accommodate operations for an execution part. The T0[OVLP] cycle can specify registers used for addressing as well. The

assumption mentioned above is also applied to the summary of cycle by cycle operation of all instructions in Appendix B and the internal operation sequences of typical instructions in Appendix C.

Addressing Parts

Because there are still more than 35 different addressing parts of the internal operations of all instructions, the addressing parts will be discussed in terms of the addressing techniques used in the 6502 for a more concise description. The addressing techniques in accordance with the addressing modes mentioned in Chapter 2 will cover the most of the addressing parts sufficiently, and are intended to analyze only the operations used for the addressing parts. In advance of analysis of any instruction, the summary of cycle by cycle operation of the instructions provided in Appendix B should be studied to be able to do cycle by cycle analysis.

The addressing part in the T0, T1, and T1[OVLP] cycles is common to all instructions. Since the discussion on the operations in these cycles has already been presented, only the operations in the T2 cycle through the T0[OVLP] cycle will be discussed as follows.

Immediate

Immediate addressing does not require any additional

cycles. Thus, the processor takes 2 cycles to perform immediate addressing. Since the instructions using the immediate addressing mode consist of two bytes, an OP CODE and a constant value, INR must be used as an addressing register that provides a 16-bit address on the IADBUS. The INR has the incremented address pointing to the location storing the OP CODE following the constant value. The reader should refer to the general sequence in Figure 4 for immediate addressing and ignore the Tn-1, Tn, and Tn-1 cycles. Note that INR will be selected among the addressing registers during the TO[OVLP] cycle.

Absolute

Absolute addressing requests at least one additional cycle. The instructions utilizing the absolute addressing mode have an OP CODE, the low order byte of an effective address, and the high order byte of the effective address. During the T1 cycle, the low order byte is fetched and

```
T2:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); IDBUS = DL; TR <= IDBUS.

T3:          IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
             RW = \1\; PC <= INR.

TO[OVLP]:    IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).
```

stored in DL. The high order byte is fetched from the consecutive location during the T2 cycle. The INR is used to address the consecutive location because INR has the

address incremented in the T1 cycle.

Before modified by the high order byte, the low order byte must be transferred from DL into TR over the IDBUS during the T2 cycle. Then, the low order byte in TR and high order byte in DL are catenated to make up a 16-bit effective address in the T3 cycle. The data to be treated in later operations will be fetched from the location specified by this address. Since the address for the next OP CODE fetch has been produced and stored in PC in the T2 and T3 cycles, PC is used as an addressing register during the T0[OVLP] cycle.

The above operations are all common to the addressing parts of the internal operations of the instructions using the absolute addressing mode except the JSR and JMP instructions. The JMP instruction uses DL and TR as addressing registers during the T0[OVLP]. The reader can ignore the R/W and SYNC outputs unless otherwise indicated. The deliberate use of PC as addressing register is arranged for the effective synthesis of the internal operations in relation to absolute indexed addressing discussed later.

The addressing of the JSR instruction requires 4 extra cycles, that is the T2 cycle through the T5 cycle. Because of the operation to store a returning address in the stack, A JSR addressing sequence will differ remarkably from

the typical absolute addressing sequence.

Before the T2 cycle, DL stores the low order byte of an effective address, and INR keeps the next consecutive address, as in the general sequence. Then, during the T2 cycle, the low order byte is transferred into TR, and the address is moved into PC. The DL and INR will become free

```
T2:          IADBUS = \0,0,0,0,0,0,0,1\,SP; DL <= DBUS;
             SYNC = \0\; RW = \1\; INR <= IADBUS; IDBUS = DL;
             TR <= IDBUS; PC <= INR.

T3:          IADBUS = INR; SYNC = \0\; RW = \0\;
             INR[8:15] <= DEC[8:15](IADBUS).

T4:          IADBUS = INR; SYNC = \0\; RW = \0\;
             INR[8:15] <= DEC[8:15](IADBUS).

T5:          IADBUS = PC; DL <= DBUS; SYNC = \0\; RW = \1\.

T0[OVLP]:    IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
             RW = \1\; INR <= INC(IADBUS).
```

for later operations. In the same cycle, the address to point to the currently used memory space in the stack is made up of an 8-bit binary one for page one and the contents of SP. Since the IDBUS is being used to transfer the low order byte into TR, a write operation is not allowed during the T2 cycle. A read operation is performed instead and the data fetched is merely discarded. The stack address is stored in INR at the end of the T2 cycle.

During the T3 and T4 cycles, the stack address in INR is decremented twice, but only the low order byte is stored in INR[8:15]. The address in PC is written on the next lower consecutive locations (2 bytes) in the stack for

a returning address. In addition, the address in PC is used to fetch the high order byte of the effective address during the T5 cycle. The location that contains the next OP CODE is addressed by DL and TR that store the high order byte and low order byte, respectively.

Zero Page

Zero page addressing requires at least the T2 cycle. The instructions using the zero page addressing mode consist of an OP CODE and an effective address in page zero, which are fetched in the T0 and T1 cycles. Now, by using the effective address in DL, the processor can address a location in page zero where either a read operation or write

```
T2:          IADBUS = \0,0,0,0,0,0,0,0\,DL; SYNC = \0\;
             RW = \0\; PC <= INR.

T0[OVLP]:    IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).
```

operation will take place. All 0's for the high order byte of a full 16-bit address is generated in the T2 cycle. All instructions with the zero page addressing mode use PC as an addressing register.

Accumulator

Accumulator addressing does not use any additional cycles like immediate addressing. However, accumulator addressing selects PC as an addressing register in place of INR because the instructions with the accumulator addressing

mode have only an OP CODE. The general sequence should be referred to for this addressing.

Implied

As far as the PHA, PHP, PLA, PLP, RTI, RTS, and BRK instructions are not concerned, the implied addressing is completely the same as accumulator addresing. No explanation will be necessary. The RTI instruction will be explained on behalf of the above instructions as follows:

As in the case of the JSR instruction, the RTI addressing employs the stack pointer. An 8-bit binary one for page one is first generated by the processor and then

```
T2:         IADBUS = \0,0,0,0,0,0,0,1\,SP; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= INC[8:15](IADBUS).

T3:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR[8:15] <= INC[8:15](IADBUS).

T4:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR[8:15] <= INC[8:15](IADBUS).

T5:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS.

T0[OVLP]:   IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
            RW = \1\; INR <= INC(IADBUS).
```

catenated with the 8-bit address stored in SP to form a stack address during the T2 cycle. Since the memory space addressed by the stack address does not contain useful data, the data to be stored in DL is not any longer used. The stack address on the IADBUS is incremented, but only the low order byte is stored in INR[8:15]. The high order byte is

directly transferred into INR[0:7]. This is for avoiding a page crossing. The previous contents of the P register are restored from the incremented stack address in the T3 cycle.

In the T3 and T4 cycles, the stack address is incremented twice to get back a 16-bit returning address. The low order byte of the returning address is first fetched and stored in DL during the T4 cycle. During the T5 cycle, while the high order byte is being fetched, the low order byte is moved into TR. Finally, an effective address from DL and TR, which is also a returning address is put on the IADBUS to perform the next OP CODE fetch. The above sequence does not include any operations for the execution part described above. A complete internal operation sequence of RTI is attached in Appendix C.

Indexed Indirect

Indexed indirect addressing requires the T2 cycle through the T5 cycle. First, placing a base address low stored in DL and eight 0's for page zero on the IADBUS, the processor fetches data during the T2 cycle. In the same cycle, the incremented address in INR is transferred into PC to be saved until it is needed. Then, INR[0:7] stores only the high order byte on the IADBUS that indicates page zero. The low order byte on the IADBUS from DL is placed on the ARGBUS for an arithmetic operation. Since the address on the IADBUS is not addressing an expected memory space in

page zero, the data fetched is simply discarded.

To get the expected address, the 8-bit data on the ARGBUS and the contents of the X register must be added in the ALU. Four arguments are required in the ALU. Argument

```
T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
            IDBUS = X; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\;
            INR[8:15] <= INC[8:15](IADBUS).

T4:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS.

T5:         IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
            RW = \1\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).
```

1 has already been placed. The X register must be connected to the IDBUS for argument 2. The rest are an operation code from the instruction decoder and two flags from the P register. A 4-bit hexadecimal zero as an operation code is used for address calculation. The two flags do not have an effect on the arithmetic operation in this case, so they are simply connected to the PS argument without any meanings. By the end of the T2 cycle, the resultant 8-bit data from the ALU is transferred into ALUREG.

During the T3 cycle, INR[0:7] and ALUREG put out a 16-bit address in page zero. The low order byte of an

effective address is fetched from the addressed location and then stored in DL. The address on the IADBUS is incremented in INC, but only the low order byte is transferred into INR[8:15]. The above operation is devised to avoid not only a page crossing but also the necessity of an 8-bit incrementer/decrementer unit in the AHPL description developed in the next chapter. The high order byte of the eff ective address is fetched from the consecutive location addressed by INR in the T4 cycle. The low order byte must be transferred into TR before deleted. Finally, DL and TR place the effective address on the IADBUS during the T5 cycle. The address of the location where the next OP CODE fetch will be performed comes from PC keeping the address since the T2 cycle.

Indirect Indexed

Indirect indexed addressing uses the Y register as an argument of the ALU for address calculation. The catenation of eight 0's and an indirect address low in DL produces a 16-bit indirect address in page zero to be used to fetch a base address low during the T2 cycle. The address of the location where the next OP CODE will be fetched is saved in PC before INR is used by another operatrion. The indirect address high is directly transferred into INR[0:7], while the indirect address low is passed through INC and then stored in INR[8:15].

While a base address-high is being fetched from the location specified by the incremented indirect address stored in INR, the base address low in DL must be added to

```
T2:        IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
           SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
           INR[8:15] <= INC[8:15](IADBUS); PC <= INR.

T3:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           OP = \0,0,0,0\; ARGBUS = Y; IDBUS = DL;
           PS = P[4],P[7];
           C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T4:        IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
           RW = \1\; INR[8:15] <= IADBUS[8:15];
           OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,1\;
           IDBUS = DL; PS = P[4],P[7];
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T5:        IADBUS = ALUREG,INR[8:15]; DL <= DBUS;
           SYNC = \0\; RW = \1\.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).
```

the data stored in the Y register. For the addition, the base address low is sent to the ALU over the IDBUS, and the contents of the Y register are placed on the ARGBUS. The operation code and the flags must be attached to the ALU as well.

The most significant bit of the outcoming 9-bit data from the ALU is stored in a flip-flop called C. The other 8 bits are put in ALUREG. If the C flip-flop is cleared, data will be fetched from the location specified by the effective address from DL and ALUREG in the T4 cycle as the last cycle. If not, the processor must execute one more arithmetic operation in the T4 cycle as follows:

First, the low order byte on the IADBUS from ALUREG must be transferred into INR[8:15] for the next cycle. The data fetched in this cycle is discarded because of an incorrect address. Then, for another arithmetic operation, the base address high currently stored in DL is put on the IDBUS and an 8-bit binary one is placed on the ARGBUS. By these operations, the base address high can be incremented by 1 in order to reflect a page crossing. In the next cycle, ALUREG and INR[8:15] produce a correct address to specify a location where read and write operations can be done. The PC as an addressing register places the address for the next OP CODE fetch on the IADBUS during the T0[OVLP] cycle.

Zero Page Indexed

Zero page indexed addressing employs the ALU for effective address calculation. The above is the only difference from zero page addressing. During the T2 cycle,

```
T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
            IDBUS = X; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR <= IADBUS.

T4:         IADBUS = INR; SYNC = \0\; RW = \0\.

T5:         IADBUS = INR; SYNC = \0\; RW = \0\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).
```

the processor places an 8-bit binary zero and the base address low kept in DL on the IADBUS to fetch data. However, the data latched into DL is completely void because the address is poniting to a false location. The primary aim of this cycle is to execute the effective address calculation in the ALU. Thus, the low order byte on the IADBUS from DL is connected to argument 1 through the ARGBUS, and the contents of either the X register or Y register are sent to argument 2 over the IDBUS.

The resultant 8-bit data from the ALU is stored in ALUREG. In the same cycle, only the IADBUS[0:7] indicating page zero is transferred into INR[0:7]. The address for the next OP CODE fetch in INR is saved in PC untill it is necessary. In the next cycle, INR[0:7] and ALUREG form an effective address. The above sequence includes two additional cycles for the instructions that perform a logic operation. The effective address during the T4 and T5 cycles is provided by INR in which the effective address on the IADBUS was stored during the T3 cycle. No page crossing is allowed, as in zero page addressing.

Indirect

Only a certain OP CODE can force the processor to perform indirect addressing. The instruction using the indirect addressing mode contains an OP CODE, an indirect address low, and an indirect address high. First, the

indirect address low in DL is transferred into TR over the IDBUS in the T2 cycle. During the same cycle, the indirect address high is fetched from the consecutive location addressed by INR and then stored in DL.

Now, DL and TR can produce a 16-bit indirect address addressing the location from which the low order byte of an effective address will be fetched during the T3 cycle. The indirect address on the IADBUS is incremented in INC and then transferred into INR. In the T4 cycle, the high order

```
T2:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             IDBUS = DL; TR <= IDBUS.

T3:          IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
             RW = \1\; INR <= INC(IADBUS).

T4:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             IDBUS = DL; TR <= IDBUS.

T0[OVLP]:    IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
             RW = \1\; INR <= INC(IADBUS).
```

byte is latched into DL by placing the address in INR on the IADBUS. Before deleted by the high order byte, the low order byte must be transferred into TR. The DL and TR will produce the effective address used for the next OP CODE fetch. Notice that the consecutive address in INR was not saved in PC during the T3 cycle because only the JMP instruction can use the indirect addressing mode.

Absolute Indexed

Unlike absolute addressing, absolute indexed addressing makes use of the ALU for effective address

calculation. In the T2 cycle, INR provides the consecutive address on the IADBUS to address the location where a base address high will be fetched. For an arithmetic operation, the base address low already stored in DL is placed on the IDBUS, and either the X register or Y register is connected to argument 1 through the ARGBUS. Operation code 0 and the flags are connected to the ALU as well. The most significant bit of the resultant 9-bit from the ALU is stored in the C flip-flop, which will indicate whether or not a page crossing is to take place. The other 8 bits are loaded into ALUREG.

If a page crossing need not occur, the T3 cycle produces an effective address by using DL and ALUREG as

```
T2:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); OP = \0,0,0,0\;
           ARGBUS = X; IDBUS = DL; PS = P[4],P[7];
           C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:        IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
           RW = \1\; INR[8:15] <= IADBUS[8:15];
           OP = \0,0,0,0\; ARGBUS = (\0,0,0,0,0,0,0,0\ !
           \0,0,0,0,0,0,0,1\) * (^C, C); IDBUS = DL;
           PS = P[4],P[7]; PC <= INR;
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T4:        IADBUS = ALUREG,INR[8:15]; SYNC = \0\; RW = \0\.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).
```

addressing registers. If a page crossing must occur, during the T3 cycle, an arithmetic operation must be performed again to increment the high order byte indicating a page number. The low order byte is saved in INR[8:15] for the

next cycle. The consecutive address in INR used for the next OP CODE fetch is also saved in PC. In the same cycle, all arguments of the ALU are set to increment the page number. As soon as the processor finishes the arithmetic operation, The resultant 8-bit data from the ALU, which is the incremented page number, is stored in ALUREG. In the T4 cycle, ALUREG and INR[8:15] are used to provide an effective address.

Relative

The instructions with the relative addressing mode consist of an OP CODE and an offset. Thus, relative addressing is exactly the same as immediate addressing as long as a branch is not taken. The general sequence in Figure 4 should be referred to in this case. A branch condition is checked in the T1 cycle. If the branch

```
T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            N <= DL[0]; OP = \0,0,0,0\;
            ARGBUS = IADBUS[8:15]; IDBUS = DL;
            PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS);
            COUT = ALU[0](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[8:15] <= IADBUS[8:15];
            OP = \0,0,0,0\; ARGBUS = (\0,0,0,0,0,0,0,1\ !
            \1,1,1,1,1,1,1,1\) * (^N & C, N & ^C);
            IDBUS = INR[0:7]; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T0[OVLP]:   IADBUS = ALUREG,INR[8:15]; IR <= DBUS;
            SYNC = \1\; RW = \1\; INR <= INC(IADBUS).
```

condition is satisfied, first the consecutive address in INR

is placed on the IADBUS for a data fetch during the T2 cycle. The data fetched is useless because the address on the IADBUS is a false branch address. For branch address calculation, the low order byte of the address on the IADBUS should be connected to argument 1, and the offset presently in DL should be connected to argument 2 in the T2 cycle.

The offset must be a value in the range from −128 to +127, and negative numbers must be expressed in the form of two's complement. The most significant bit of the offset in DL is saved in a flip-flop named N to prepare for the next cycle aimed at a page crossing. The N flip-flop will be used to indicate whether the offset is a positive number or negative number during the T3 cycle. By the end of the T2 cycle, the most signifiacant bit of the resultant 9-bit data from the ALU is stored in the C flip-flop, and the other 8 bits are stored in ALUREG as before.

To predict if the page should be crossed or not, during the T2 cycle, the most significant bit of the data from the ALU must be directly connected to a communication terminal called COUT. The reason of the above operation is that the C flip-flop can not output its true value during the T2 cycle. The most significant bit of the offset in DL (DL[0]) is also devised for the same reason.

If both the COUT and DL[0] are set or cleared, no page crossing occurs. Thus, the next cycle becomes the next

OP CODE fetch cycle for which the branch address is prepared by INR[0:7] containing the original page number and ALUREG including the low order byte of the branch address. If the above condition is not satisfied, another arithmetic operation must be performed in the T3 cycle. First, the low order byte on the IADBUS is transferred into INR[8:15]. The INR[0:7] is connected to argument 2 through the IDBUS. According as what values the C and N flip-flops are storing, either an 8-bit binary 1 or 8-bit binary 255 implying -1 in two's complement form is connected to argument 1 through the ARGBUS. The resultant 8-bit data from the ALU is transferred into ALUREG. Now, ALUREG and INR[8:15] produce the correct branch address pointing to the location from which the next OP CODE will be fetched.

By now, each addressing technique has been discussed by using a typical addressing sequence of each addressing technique. However, the typical sequence of a addressing technique is not necessarily applicable to the addressing parts of all instructions using the same addressing technique. Some instructions may require more cycles to accomplish their particular addressing, as in the JSR and RTI addressing. In fact, an addressing sequence of an instruction can be easily derived by means of the corresponding typical sequence. The addressing registers that provide a 16-bit address on the IADBUS for the next

OP CODE fetch cycle is summarized in Table 3. It will help to construct an internal operation sequence for each instruction.

Execution Parts

Most of the instructions force the processor to execute not only the operations for an addressing part but also the operations for an execution part. Only the JMP and branch instructions do not have an execution part. Since the 6502 specifies 56 kinds of instructions, it will be a nuisance to describe the execution parts of all these instructions. Furthermore, the cycles in which the execution part of an instruction is performed are not fixed, and are changed depending on the addressing mode the instruction is using. For the above reasons, the execution parts will be discussed in accordance with the fundamental instructions explained in Chapter 2.

Arithmetic

The execution parts that require an arithmetic operation must employ the ALU. All capabilities necesary for carrying out the execution parts associated with an arithmetic operation have been provided in the ALU. Therefore, only connections to the four arguments of the ALU must be properly assigned. The specification of the arithmetic-logic unit employed in a 6502 AHPL description

Table 3. Summary of the Addressing Registers

Instructions	Addressing Mode	Cycle #	Registers used for addressing
ADC AND CMP CPX CPY EOR LDA LDX LDY ORA SBC	Immediate	2	INR
JMP	Absolute	3	DL,TR
ADC AND BIT CMP CPX CPY EOR LDA LDX LDY ORA SBC	Absolute	4	PC
STA STX STY	Absolute	4	PC
ASL DEC INC LSR ROL ROR	Absolute	6	PC
JSR	Absolute	6	DL,TR
ADC AND BIT CMP CPX CPY EOR LDA LDX LDY ORA SBC	Zero Page	3	PC
STA STX STY	Zero Page	3	PC
ASL DEC INC LSR ROL ROR	Zero Page	5	PC
ASL LSR ROL ROR	Accumulator	2	PC
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	Implied	2	PC
PHA PHP	Implied	3	PC
PLA PLP	Implied	4	PC
RTI	Implied	6	DL,TR

Table 3. -- Continued

Instructions	Addressing Mode	Cycle #	Registers used for addressing
RTS	Implied	6	INR
BRK	Implied	7	DL,TR
ADC AND CMP EOR LDA ORA SBC	Indexed Indirect	6	PC
STA	Indexed Indirect	6	PC
ADC AND CMP EOR LDA ORA SBC	Indirect Indexed	5	PC
ADC AND CMP EOR LDA ORA SBC	Indirect Indexed	6*	PC
STA	Indirect Indexed	6	PC
ADC AND CMP EOR LDA LDY ORA SBC	Zero Page Indexed,X	4	PC
STA STY	Zero Page Indexed,X	4	PC
ASL DEC INC LSR ROL ROR	Zero Page Indexed,X	6	PC
LDX	Zero Page Indexed,Y	4	PC
STX	Zero Page Indexed,Y	4	PC
JMP	Indirect	5	DL,TR
ADC AND CMP EOR LDA LDY ORA SBC	Absolute Indexed,X	4	PC
ADC AND CMP EOR LDA LDY ORA SBC	Absolute Indexed,X	5*	PC
STA	Absolute Indexed,X	5	PC

* --- With Page Crossing.

Table 3. -- Continued

Instructions	Addressing Mode	Cycle #	Registers used for addressing
ASL DEC INC LSR ROL ROR	Absolute Indexed,X	7	PC
ADC AND CMP EOR LDA LDX ORA SBC	Absolute Indexed,Y	4	PC
ADC AND CMP EOR LDA LDX ORA SBC	Absolute Indexed,Y	5*	PC
STA	Absolute Indexed,Y	5	PC
BCC BCS BEQ BMI BNE BPL BVC BVS	Relative	2@	INR
BCC BCS BEQ BMI BNE BPL BVC BVS	Relative	3$	INR[0:7],ALUREG
BCC BCS BEQ BMI BNE BPL BVC BVS	Relative	4%	ALUREG,INR[8:15]

* --- With Page Crossing. @ --- Branch Not Taken.
$ --- Branch Taken Without Page Crossing.
% --- Branch Taken With Page Crossing.

that describes the operation codes, the instructions and the operations using an arithmetic or logic operation, and the expressions indicating the results of all kinds of arithmetic and logic operations is shown in Table 4.

The arithmetic operation of binary addition without the carry input is specified by operation code 0. This arithmetic operation is intended for the execution parts of the DEC, DEX, DEY, INC, INX, and INY instrucions. First, the data to be processed must be fetched from the location specified by operations for an addressing part in the case of DEC and INC. For the instructions that decrement the data in DL, X, or Y, the processor has to generate eight 1's implying -1 in two's complement form and place them on the ARGBUS connecting to argument 1. For the instructions that increment the data in DL, X, or Y, an 8-bit binary 1 must be put on the ARGBUS connecting to argument 1. The data to be processed must be placed on the IDBUS connecting to argument 2. The carry and decimal mode flags in the P register are merely connected to the PS argument, since they do not have any effects on the arithmetic operation. This arithmetic operation is also utilized for address calculation.

For the ADC instruction, the arithmetic operation by which argument 1, argument 2, and the carry input are added together is selected by operation code 1. The carry flag in the P register, as the carry input, must be connected to

Table 4. Specification of the Arithmetic-Logic Unit

OP Code	Instructions or Operations that use ALU operation.	Internal Operation ALU(OP; ARG1; ARG2; PS)
0	Address Calculation, Load, DEC, DEY, DEX, INC, INX, and INY	ARG1 + ARG2
1	ADC(if PS[0]=0, binary ADC; otherwise decimal ADC.)	ARG1 + ARG2 + PS[1]
2	AND and BIT	\0\,ARG1 & ARG2
3	ORA	\0\,ARG1 V ARG2
4	EOR	\0\,ARG1 @ ARG2
5	SBC(if PS[0]=0, binary SBC; otherwise decimal SBC.)	ARG1 - ARG2 - $\overline{\text{PS[1]}}$
6	CMP, CPX, and CPY	ARG1 - ARG2
7	ASL	ARG2,\0\
8	LSR	ARG2[7],\0\,ARG2[0:6]
9	ROL	ARG2,PS[1]
A	ROR	ARG2[7],PS[1],ARG2[0:6]

+: Add   -: Subtract   &: AND   V: OR   @: Exclusive OR

PS[1], and the decimal mode is also connected to PS[0] for the arithmetic operation. Both of the PS argument have an effect on the arithmetic operation. If PS[0] is cleared, binary addition will be performed, otherwise decimal addition will be performed. The AC is always connected to argument 1, while DL is always connected to argument 2.

The arithmetic operation of subtraction with borrow is specified by operation code 5. The carry and decimal flags have effects on the arithmetic operation, as in the arithmetic operation for ADC. However, the carry flag is used as the borrow in this case. No negation of PS[1] is required because the ALU handles such an operation in itself. The data to be subtracted must be connected to argument 1, while the data to subtract must be connected to argument 2. Since only the SBC instruction can use this arithmetic operation, AC and DL are always connected to argument 1 and argument 2, respectively.

The arithmetic operation of subtraction without borrow is designed for the CMP, CPX, and CPY instructions. Operation code 6 can specify this arithmetic operation. Unlike the arithmetic operation for SBC, the PS argument does not affect the arithmetic operation at all. Argument 1 and argument 2 must be connected in the same manner taken for SBC. The AC, X, or Y is connected to argument 1 through the ARGBUS, and DL storing data from memomy is connected to

argument 2 through the IDBUS.

Once connections to the ALU have been defined by the operations for an execution part, the ALU will begin to produce 9-bit data. The 8 bits of the resultant data exclusive of the most significant bit are stored in ALUREG temporarily, and then transferred into one of the internal registers. The selection of a destination register must also be done. So far, an arithmetic operation has been performed, and the resultant data from the ALU has been stored in a destination register. The rest of the operations are used to refresh the P register. In order to renew the zero flag, the 8 bits exclusive of the most significant bit are ORed bit by bit, and then the resultant bit is complemented. If the bit is a 1, the data is equal to 0, otherwise the data has a value other than 0.

The most significant bits of the orignal data placed on the IDBUS and ARGBUS as well as the most significan bit of the resultant data are compared to see if an overflow condition was generated as a result of the arithmetic operation. The overflow flag is set when an overflow condition occured. Only the execution parts of ADC and SBC have the operations renew the overflow flag. The negative and carry flags can be directly set or cleared by the corresponding bits of the 9-bit resultant data. Because the P register must be refreshed by the end of the TO[OVLP]

cycle, the data stored in ALUREG can not be used. In fact, some instructins request an arithmetic operation during the TO[OVLP] cycle.

Logic

The ALU must also perform logic operations for the functions of AND, OR, excusive OR, shift, and rotation. The functions of shift and rotation are classified into a logic operation for compact discussion, as mentioned before. Only the two types of logic operations for the rotation function use the PS argument, though all logic operations use argument 2. Only the logic operations for the AND, BIT, ORA, and EOR instructions use argument 1. For the logic operations to generate the AND, OR, and excusive OR functions, AC is connected to argument 1 through the ARGBUS, while DL is connected to argument 2 through the IDBUS. For the logic operations for the shift and rotation functions, the data in AC is placed on the ARGBUS connecting to argument 1. The operation codes for the respective logic operations should be refferred to in the Table 4.

Once a logic operation has been performed, the 8 bits of the resultant 9-bit data exclusive of the most significant bit will be stored in ALUREG. The most significant bit is used to renew the carry flag for such instructions as ASL, LSR, ROL, and ROR. Data transfer from ALUREG into AC must be defined for all the instructions

except BIT. all instructions associated with a logic operation refresh the zero and negative flags. However, the BIT instruction must renews the negative and overflow flags by using the most significant bit and the next most significant bit of DL, respectively.

Load

The load instruction takes advantage of the arithmetic operation specified by operation code 0. The reason of use of the arithmetic operation is that only the data from the ALU is allowed to affect the P register except for restoring the previous contents of the P register from the stack. Therefore, the data latched into DL must first go through the ALU. Since the data is not supposed to be changed, zero should be added to the data in the ALU. The carry and zero flags are cleared or set in the same way discussed before.

For the arithmetic operation, DL must be connected to argument 1 through the IDBUS, and eight 0's must be placed on the ARGBUS connecting to argument 2. The cycle in which the arithmetic operation is performed will be determined by the addressing mode that an instruction, which will be one of the LDA, LDX, and LDY instructions, is using. Once the arithmetic operation has been performed, the resultant 8-bit data is stored in ALUREG and then transferred into one of the destination registers, AC, X,

and Y.

Store

The only operation is to place the data stored in AC, X, or Y on the IDBUS in an appropriate cycle. Note that logic 0 must be placed on the SYNC line in the same cycle to indicate the execution of a write operation. The IDBUS must be explicitly connected to the DBUS.

Jump

The BRK, JMP, JSR, RTI, and RTS instructions are jump instructions. As already stated, The JMP does not have an execution part. All operations that JMP must do are used for its addressing part. The other four instructions have operations to either store data in the stack or load data from the stack. Since they use only the implied addressing mode, the cycles in which the operations for their execution parts will be done are fixed. The operations for JSR will be explained on behalf of the se four instructions as follows:

The first cycle in which the processor can execute some operations for an execution part is the T2 cycle. The first operation of JSR is a write operation in the stack. However, a write operation is not allowed in the T2 cycle, because the IDBUS is used for a data transfer from DL into TR. Thus, the write operation to store the high order byte

of a returning address stored in PC must wait for the T3 cycle. The address pointing to the currently used location in the stack has been produced during the T2 cycle.

In the T3 cycle, the contents of PC[0:7], the upper half byte of the returning address, are placed on the IDBUS and then written in the location through the data bus. The IDBUS is explicitly connected to the DBUS because the connection is cycle-dependent. During the T4 cycle, the low order byte of the returning address must be written in the next lower location. The address addressing the location has been produced and stored in INR by the beginning of the T4 cycle. Since the returning address has been saved in the stack by the beginning of the T5 cycle, no write or read operation is expected.

The remaining operations for the execution part are to set the new stack address in SP so that the next read or write operation in the stack can be done correctly. During the T5 cycle, the latest stack address is transferred from INR into PC while the high order byte of a jump address is being fetched. The next cycle is the next OP CODE fetch cycle, but some operations can be overlapped as long as the conflict on the internal buses, internal registers, and CLU's does not occur. Since the IDBUS is free during the T0[OVLP] cycle, the contents of PC[8:15] can be transferred into SP over the IDBUS. The page number of the stack is

always page 1; therefore, only the low order byte is stored in SP. In fact, SP is an 8-bit register. No overlap is necessary during the T1[OVLP] cycle because all operations have been completed by the end of the T0[OVLP].

An internal operation sequence of JSR has been provided in Appendix C. Appendices A and B should be referred to before analyzing any instruction.

Set/Clear

All instructions classified into the set/clear instruction use only the implied addressing mode. Furthermore, the instructions require the same number of cycles. So the operations for the execution parts of the set/clear instructions as well as the addressing parts can be uniformed except the flags to be set or cleared by the in structions. Since the instructions do not employ any additional cycles, the execution parts must be difined within the T0[OVLP] and T1[OVLP] cycles. The only operation to set or clear the corresponding flag for each execution part is to be done. Thus, only one cycle is necessary to carry out the above operation. T0[OVLP] cycle is selected because of the fact that the P register is checked by the branch instructions in the T1[OVLP] cycle.

Transfer

Like the set/clear instructions, all transfer

instructions require no additional cycles and utilize the implied addressing modes. However, they must use both the T0[OVLP] and T1[OVLP] cycles for their execution parts.

In the T1[OVLP] cycle, the data to be transferred must first be put in the ALU and then stored in ALUREG. The reason of the above operation is that only the data from the ALU can affect the P register. Therefore, the negative and zero flags will be set or cleared depending on the data from the ALU. Operation code 0 specifies the arithmetic operation used for this operation. One of the source registers, AC, X, Y, and SP must be connected to argument 2, while the an 8-bit binary 0 must be placed on the ARGBUS connecting to argument 1. Note that although the TXS does not affect any flags, the data stored in X is transferred into SP through the ALU and ALUREG for the uniformity of the operations designed for the transfer instructions. During the T1[OVLP] cycle, the data in ALUREG is routed into one of the destination registers, AC, X, Y, and SP. A source and destination registers are determined by a transfer instruction.

Stack

The stack instructions can be further divided into push instructions such as PHA and PHP and pull instructions such as PLA and PLP. The execution parts of the pull instructions will be analyzed as follows:

During the T2 cycle, a stack address is provided on the IADBUS by using SP. Either the contents of AC or the contents of P are placed on the IDBUS and then written in the location addressed by the stack address. Since the IDBUS is not used for another operation necessary for later operations, the T2 cycle can be effectively used for the write operation. In the same cycle, the stack address is decremented to keep track of the currently used stack address. The address is saved in PC in the T0[OVLP] cycle, and stored in SP during the T1[OVLP] cycle.

The execution parts of the pull instructions require an additional cycle because the location specified by SP does not contain expected data. So, the data fetched in the T2 cycle is merely discarded. The next upper stack address is obtained by incrementing the stack address on the IADBUS. Now, the expected data from the location pointed to by the incremented address in INR can be fetched during the T3 cycle. In the T0[OVLP] cycle, the fetched data in DL is transferred into the P register for PLP or into the ALU for PLA. The data from the ALU is used to refresh the negative and zero flags. Because the data in the P register affected itself directly, no transfer to the ALU is assumed in the case of PLP. The stack pointer is refreshed in the same way the pull instructions use.

The internal operations of all instructions have

been analyzed in terms of the addressing parts and execution parts. Internal operation sequences of 37 instructions have been attached in Appendix C in accordance with the 37 kinds of addressing patrts employed in the 6502. The internal operation of any instruction can be analyzed by means of the corresponding sequence. The 37 internal operation sequences will help to develop a 6502 AHPL description very effectively. So far, no conflict between the interanl architecture shown in Figure 2 and the internal operation sequences appears. The reader should note that the internal operation sequences are not exactly same as, what is called, a control sequence in reference 1. They do not have step numbers and a branch part.

Inputs

Since the internal operations of all instructions have been discussed by this point, the internal operations of all inputs should next be analyzed in order to complete analysis of the whole internal operations of the 6502. Analysis of the outputs will not be necessary because they are provided to reflect the results of the internal operations. There are six inputs, that is, $\overline{IRQ}$, $\overline{NMI}$, $\overline{RES}$, RDY, PHIO, and S.O. in the 6502. No explanation of PHIO will be required because of its charcteristic. First, the $\overline{IRQ}$, $\overline{NMI}$, and $\overline{RES}$ inputs, which have rather complicated operations will be analyzed. The RDY and S.O. inputs will

be discussed next at the conclusion of this chapter.

Interrupt Inputs

The 6502 has two types of interrupt inputs. The $\overline{IRQ}$ input is a maskable interrupt input, while the $\overline{NMI}$ input is a non-maskable interrupt input. Therefore, the $\overline{NMI}$ input has priority over the $\overline{IRQ}$ input. The $\overline{RES}$ input is not called an interrupt input. However, the $\overline{RES}$ input will be analyzed here together with the $\overline{IRQ}$ and $\overline{NMI}$ inputs because the internal operation of the $\overline{IRQ}$ input is quite similar to those of the $\overline{IRQ}$ and $\overline{NMI}$ inputs. The $\overline{RES}$ input can be thought as an interrupt input. By using the general sequnce of the interrupt inputs given in Figure 5, the internal operations of the interrupt inputs will be analyzed cycle by cycle as follows:

T0 -- The T0 cycle is the OP CODE fetch cycle, but the OP CODE fetched is void since the processor is about to execute the internal operation of an interrupt input, one of the $\overline{RES}$, $\overline{IRQ}$, and $\overline{NMI}$ inputs. Like the internal operation of an instruction, the internal operation of an interrupt starts with the T0 cycle. The T0 cycle is simply furnished in conjunction with the internal operations of the instructions. The address on the IADBUS is tranferred into INR either directly for $\overline{IRQ}$ and $\overline{NMI}$ or through INC for $\overline{RES}$. For $\overline{RES}$, the incremented address in INR does not have any meanings. On the other hand, the non-incremented address in

T0: IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;

INR <= IADBUS. for $\overline{\text{RES}}$.
INC(IADBUS). for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

T1: IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
INR <= INC(IADBUS); PC <= INR.

T2: IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;

RW = \1\; for $\overline{\text{RES}}$.
\0\; for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

IDBUS = PC[0:7];
DBUS = IDBUS; for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

INR[0:7] <= IADBUS[0:7];
INR[8:15] <= DEC[8:15](IADBUS).

T3: IADBUS = INR; SYNC = \0\;

RW = \1\; for $\overline{\text{RES}}$.
\0\; for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

IDBUS = PC[8:15];
DBUS = IDBUS; for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

P[3] <= \0\;
for $\overline{\text{IRQ}}$.

INR[8:15] <= DEC[8:15](IADBUS).

T4: IADBUS = INR; SYNC = \0\;

RW = \1\; for $\overline{\text{RES}}$.
\0\; for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

IDBUS = P;
DBUS = IDBUS; for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

INR[8:15] <= DEC[8:15](IADBUS).

Figure 5. General Sequence for Interrupt Inputs

T5: IADBUS =

\1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0\; for $\overline{\text{RES}}$.
\1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0\; for $\overline{\text{IRQ}}$.
\1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0\; for $\overline{\text{NMI}}$.

DL <= DBUS; SYNC = \0\; RW = \1\;
INR <= INC(IADBUS); PC <= INR.

T6: IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
IDBUS = DL; TR <= IDBUS; P[5] <= \1\.

T0[OVLP]: IDBUS = DL,TR; IR <= DBUS; SYNC <= \1\;
RW = \1\; INR <= INC(IADBUS);
IDBUS = PC[8:15]; SP <= IDBUS.

Figure 5. -- Continued

INR is directly related to a returning address to be stored in the stack for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$.

T1 -- The T1 cycle is the data fetch cycle following the OP CODE fetch cycle. However, the data fetched in this cycle is totally useless. The operation of the address transfer from INR into PC is only important to the interrupt inputs. The other operations are merely provided in relation to the internal operations of the instructions.

T2 -- The stack address on the IADBUS is produced by catenating an 8-bit binary one and the contents of SP. For $\overline{\text{RES}}$, a read operation is performed, but no data is required to be stored. The T2 cycle is an idle cycle for $\overline{\text{RES}}$. The high order byte of a returnig address in PC is saved in the location specified by the stack address on the IADBUS for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$. The R/W line must be pulled either high or low according to an interrupt input. Then, the high order byte of the stack address indicating the page number is transferred into INR[0:7]. For this operation, the processor will not need to generate an 8-bit binary one for page one any longer. The low order byte is transferred into DEC and then stored in INR[8:15].

T3 -- For $\overline{\text{RES}}$, data is fetched from the next lower location by using INR. The data is simply discarded. For $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$, the low order byte of the returning address is written in the location. The stack address on the IADBUS is

decremented again during the T3 cycle. Note that the break command flag is cleared to indicate the acceptance of an $\overline{IRQ}$ interrupt input signal. No such operation needs to take place in the internal operations of $\overline{RES}$ and $\overline{NMI}$. This is because that the BRK instruction and $\overline{IRQ}$ input use the same vector locations FFFE and FFFF. The stack address is decremented and stored in INR, as in the T2 cycle.

T4 -- The operartions in the T4 cycle are almost the same as those in the T2 and T3 cycles. A main difference is that the contents of the P register are saved in the stack for $\overline{IRQ}$ and $\overline{NMI}$. A read operation is performed again for $\overline{RES}$.

T5 -- The processor generates the vector address, FFFA for $\overline{NMI}$, FFFC for $\overline{RES}$, or FFFE for $\overline{IRQ}$ and then places it on the IADBUS. The vector address on the IADBUS is incremented in INC and then stored in INR to prepare for the next data fetch.

The data fetched is the low order byte of the starting address of an interrupt routine. The latest stack address in INR is transferred into PC before modified by the next vector address incremented in INC.

T6 -- During the T6 cycle, the high order byte of the starting address is fetched from the location specified by INR. The low order byte stored in DL is transferred into TR over the IDBUS before deleted by the high order byte. In

the same cycle, the interrupt disable flag is set to reject further interrupt signals. However, the flag can not disable the $\overline{RES}$ and $\overline{NMI}$ input signals.

TO[OVLP] -- The DL and TR produce the address to point to the location where the first data fetch of the interrupt routine will be done. The low order byte of the stack address in PC is sent to SP over the IDBUS.

Complete internal operation sequences of the $\overline{RES}$, $\overline{IRQ}$, and $\overline{NMI}$ inputs have been attached in Appendix D.

Simple Inputs

There are two inputs yet to be analyzed. The RDY and S.O. inputs have rather simple operations than the interrupt inputs discussed right before.

RDY -- As briefly described in Chapter 2, the RDY input can halt all cycles except write cycles. In the context of digital hardware systems, the term, halt usually implies the halt of clock signals used for registers, flip-flops, etc. Thus, the signal generated by RDY must be connected to a clock generator in order to be able to stop the transitions of clock. To generate a control signal, The RDY must be conditioned with the R/W output because only read cycles can be halted. The exact operations of RDY will be discussed in the next chapter in association with a 6502 AHPL description.

S.O. -- The operation of the S.O. input can be

implemented by one transfer operation. When the S.O. input is pulled low, the overflag will be set immediately.

Internal operation sequences of these inputs have also been shown in Appendix D.

CHAPTER 4

AHPL DESCRIPTION

In order to realize the 6502 microprocessor in the form of an SLA, a 6502 AHPL description must be developed first. For the development, the internal operations of the 6502 have been analyzed by now. An efficient 6502 AHPL description is requested because of the nature of a VLSI implementation. For this reason, internal operation sequences of typical instructions have been produced. It will help to synthesize the internal operations effectively.

The purpose of this chapter is to develop a 6502 AHPL description aimed at an SLA realization step by step. This chapter will also discuss the methods utilized in developing the 6502 AHPL description. In this chapter, it is assumed that the reader has an enough knowledge of AHPL. A reader who does not have an experience of designing a hardware system by using AHPL should refer to reference 1 before getting into the following discussion. The reader shall often see the 6502 AHPL description provided in Appendix F as the discussion in this chapter proceeds.

Declarations

As the first step, a module name, the internal

buses, the internal registers, the CLU's, the inputs, the outputs, the buses, control flip-flops, and communication terminals must be properly declared in developing a 6502 AHPL description. All internal registers of the 6502 as well as several control flip-flops are declared in a MEMORY section. The six inputs are declared in an EXINPUTS section, while the outputs are declared in an OUTPUTS section. The address bus is also placed in the OUTPUTS section. Some communication terminals must be declared in the OUTPUTS section as well. A BUSES section defines the internal buses, while an EXBUSES section defines the data bus. According to a rule defined for the new multi-stage compiler, the three CLU's are declared in three CLUNITS sections separately.

Note that the INC and DEC units are declared separately, though the internal achitecture in Figure 2 has the INC/DEC unit. The use of the INC and DEC units instead of the INC/DEC unit will be justified by the simplicity of the 6502 AHPL description. Explanations of those control flip-flops and communication terminals will be given when necessary.

Data Fetch Cycle

As stated in the preceding chapter, the data fetch cycle following the OP CODE fetch cycle is common to all internal operations. The common data fetch cycle is defined

in step 2. Before the discussion of step 2, step 1 must be developed first.

S1 -- Since step 1 is to be associated with the CONTROLRESET(1) statement, step 1 must define the T0 cycle of the internal operation of the $\overline{\text{RES}}$ input. The above comes from the fact that the 6502 is started or reset by a signal on the $\overline{\text{RES}}$ input line. Thus, the compiler must understand that step 1 will be executed immediately when the RES input declared in the EXINPUTS section went low. The above capability has been provided in the new compiler. However, the statement for that purpose is not inserted in the 6502 AHPL description attached in Appendix F. The grammer concerning CONTROLRESET(1) had not been available by the time the latest version of the AHPL description was completed. A read operation is merely performed in step 1. In addition, the control flip-flops, IRQF and NMIF that will indicate the acknowledgement of $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ input signals are cleared. The necessity of these operations will be clarified in later steps.

S2 -- Step 2 is for the common OP CODE fetch cycle. The control flip-flop, EXT is used to indicate whether or not one of the internal operations of the $\overline{\text{RES}}$, $\overline{\text{IRQ}}$, and $\overline{\text{NMI}}$ inputs is to be performed. If one of the control flip-flops, RESF, IRQF, and NMIF has been set, the EXT flip-flop will be set and control branches to step 25 to

execute the rest of the internal operation of the interrupt input whose control flip-flop has been set. If not, control continues to step 3 to decode the OP CODE currently stored in the IR. The other operations in step 2 have been explained in Chapter 3.

Instruction Decoding

Before the common data fetch cycle in step 2, an OP CODE has been stored in IR by one of steps 21, 22, 23, and 24 intended for the OP CODE fetch cycle, except for the initial operation by the $\overline{\text{RES}}$ input. Instruction decoding must take place in the internal operations of all instructions. To determine how many additional cycles the instruction being stored in IR will require, the instruction decoding must be done within the data fetch cycle.

It is not efficient to write one big conditional branch statement in step 2 because of the 151 8-bit OP CODE's. Furthermore, it is not desirable to have individual steps for the internal operation of each instruction since each non-NODELAY step requires one D flip-flop for sequence control. On the other hand, it will be clumsy to have only one step for the same cycle of the internal operations of all instructions and inputs because of the necessity of a large number of condition parameters to operate proper conditional transfers, connections, and branches. A moderate number of steps should be used to

accomodate the entire internal operations so that conditional transfers, connections, and branches will need a moderate number of condition parameters.

Common operational features have been checked by means of the internal operation sequences in Appendix C. As the result, related instructions and inputs are classified into six groups called scheme A, B, C, D, E, and F. The classification of the instructions and inputs is neatly indicated in Table 5. A Karnaugh map method will be extensively utilized for not only the instruction decoding but also the synthises of the internal operations. A Karnaugh map of the operation codes has been provided in Appendix E for readers' covenience. Step 3 through step 20 associated with the instruction decoding will be discussed as follows:

S3 -- Since 256 OP CODE's can be defined by the combination of 8-bit data, 105 undefined OP CODE's on the map attached in Appendix E must be eliminated to identify one of the scheme which the OP CODE stroed in IR belongs to. If the OP CODE is one of the OP CODE's in the area confined by IR[6]IR[7] = 11 on the map, control jumps to step 21 in which the next OP CODE fetch will be performed. The 64 undefined OP CODE's on the map are eliminated by the condition parameters IR[6] and IR[7]. An undefined OP CODE is regarded as a non-operation code in the AHPL description.

Table 5. Classification of Instructions by Addressing Modes and Features

Scheme	Instructions	Common Features
A	PHA PHP PLA PLP RTS RTI BRK	Stack related Instructions and Operations.
	JSR	
	$\overline{NMI}$ $\overline{IRQ}$ $\overline{RES}$(Operations)	
B	ASL LSR ROL ROR	Accumulator and Implied Mode Instructions (2 cycles).
	CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXS TXA TYA	
C	ADC AND CMP CPX CPY EOR LDA LDX LDY ORA SBC	Immediate and Relative Mode Instructions (2 bytes).
	BCC BCS BEQ BMI BNE BPL BVC BVS	
D	JMP	Absolute Mode Instructions.
	ADC AND BIT CMP CPX CPY EOR LDA LDX LDY ORA SBC	
	STA STX STY	
	ASL DEC INC LSR ROL ROR	
	ADC AND CMP EOR LDA LDY ORA SBC (Index X)	Absolute Indexed Mode Instructions.
	STA (Index X)	
	ASL DEC INC LSR ROL ROR (Index X)	

Table 5. -- Continued

Scheme	Instructions	Common Features
D	ADC AND CMP EOR LDA LDX ORA SBC (Index Y)	Absolute Indexed Mode Instructions.
	STA (Index Y)	
	JMP	Indirect Mode Instruction.
E	ADC AND BIT CMP CPX CPY EOR LDA LDX LDY ORA SBC	Zero Page Mode Instructions.
	STA STX STY	
	ASL DEC INC LSR ROL ROR	
	ADC AND CMP EOR LDA LDY ORA SBC (Index X)	Zero Page Indexed Mode Instructions.
	STA STY (Index X)	
	ASL DEC INC LSR ROL ROR (Index X)	
	LDX (Index Y)	
	STX (Index Y)	
F	ADC AND CMP EOR LDA ORA SBC	Indexed Indirect Mode Instructions.
	STA	
	ADC AND CMP EOR LDA ORA SBC	Indirect Indexed Mode Instructions.
	STA	

In other words, all undefined OP CODE's are treated like the NOP instruction. No information with respect to an undefined OP CODE was available at the time of the development. Since the instruction decoding must be completed by the beginning of the next operation cycle, a NODELAY statement is put in each step used for the instruction decoding. By using a NODELAY statement, the instruction decoding is not necessarily placed in step 2. A NODELAY step will not harm a hardware realization a lot because no flip-flop is required for sequence control. If the OP CODE in IR has not been selected by the condition, control continues to step 4. Note that thare are still more than 40 undefined OP CODE's on the map.

S4 -- The instructions grouped into scheme A such as BRK, JSR, PHA, PHP, PLA, PLP, RTI, and RTS are checked in step 4. If the OP CODE matches one of these instructions, control branches to step 25. In this case, the instruction decoding is not required any longer. The further distinction of those instructions will be done by using conditional transfers, connections, and branches in the steps concerned with scheme A. If not, control advances to step 5.

S5 -- By the condition IR[4] & ^IR[5] & ^IR[7], step 5 checks whether or not the OP CODE is for one of the instructions classified into scheme B. If so, control jumps

to step 10, otherwise control goes down to the next step. Note that the area confined by the condition still has six undefined OP CODE's. The condition parameters will select sufficiently only the scheme B instructions because the PHA, PHP, PLA, and PLP instructions that are also in the area confined by the condition have been eliminated in step 4.

S6 -- Step 6 selects the instructions employing the immediate or relative addressing mode placed under scheme C. The area confined still includes undefined OP CODE's. The branch condition in step 6 is shown in step 6 of the AHPL description in Appendix F. If the confition is satisfied, control branches to step 12 for further elimination of the undefined OP CODE's, otherwise control continues to step 7.

S7 -- By the condition IR[4] = 1, the instructions using the absolute, absolute indexed, or indirect addressing mode can be selected. They are categorized into scheme D. The above simple condition is because the OP CODE's in the area confined by IR[4]IR[5] = 10 have already been eliminated in the previous steps. If the condition is true, then control jumps to step 16. If false, control continues to step 8. The area confined still contains undifined OP CODE's.

S8 -- Step 8 determines whether or not the OP CODE currently stored in IR is for one of the instructions utilizing the indexed indirect or indirect indexed

addressing mode. Only IR[5] is used as a branch condition. If IR[5] of the OP CODE is equal to 1, control branches to step 19 for further decoding, otherwise control advances to step 9 for scheme F.

S9 -- Since the OP CODE not selected in step 3 through step 8 is for a scheme E instruction, control jumps unconditionally to step 42 where the T2 cycle of the internal operation of the scheme E instruction will be performed. No elimination of undefined OP CODE's is necessary. This step can be omitted by adding the negative condition ^IR[5] to the branch statement of step 8.

S10 -- Step 10 is used to remove the undefined OP CODE's from the OP CODE's selected in step 5. If the OP CODE currently stored in IR is one of the instructions classified into scheme B, control proceeds to step 11. In the case that the OP CODE matches one of the undefined OP CODE's or the NOP instruction, control branches to step 21 as usually. The NOP is removed in step 10 to make the description for the internal operations of the scheme B instructions simpler.

S11 -- To avoid writing a lengthy conditional branch statement in step 10, step 11 is added. Step 11 provides an uncoditional branch to step 30 where the T2 cycle of the internal operations of the scheme B instructions will be performed.

S12 -- Step 12 removes undefined OP CODE's from the OP CODE's selected in step 6. If the OP CODE in IR is one of the undefined OP CODE's, control branches to step 21 for the next OP CODE fetch, otherwise control continues to step 13.

S13 -- The OP CODE's not chosen in the preceding step consist of the OP CODE's for the scheme C instructions and several undefined OP CODE's. Some of the defined OP CODE's are checked in step 13 by the condition IR[0] & ^IR[1] & IR[2] & ^IR[3]. If the condition is satisfied, control jumps to step 31 for the internal operation of a scheme C instruction, otherwise control goes down to step 14.

S14 -- Step 14 is also used to choose the undefined OP CODE's from the OP CODE's not selected in step 13. For the undefined OP CODE's, control branches to step 21. For the OP CODE's for the remaining scheme C instructions, control advances to the next step. The last three steps, S12 , S13, and S14 are devised for simple branch conditions as well as decoding functions.

S15 -- Control branches unconditionally to step 31 to perform the internal operation of a scheme C instruction.

S16 -- Step 16 is provided to select some of the undefined OP CODE's from the OP CODE's chosen in step 7. Control jumps to step 21 for the undefined OP CODE's or goes

to step 17 for the OP CODE's not selected.

S17 -- The area not confined in the preceding step still has some undefined OP CODE's. If the branch condition is true, the OP CODE in IR is undefined, otherwise the OP CODE is for one of the scheme D instructions. Control branches to step 21 for the undefined OP CODE's or advances to the next step for defined OP CODE's.

S18 -- Step 18 is provided to make the branch condition of step 17 concise. Control jumps unconditionally to step 35 for the T2 cycle of the internal operation of a scheme D instruction.

S19 -- Step 19 removes all undefined OP CODE's from the OP CODE's selected in step 8. Control advances to step 20 if the OP CODE is for one of the scheme E instructions.

S20 -- Once control has reached step 20, control branches unconditionally to step 40 to start the T2 cycle of the internal operation of a scheme E instruction.

A flowchart for the instruction decoding together with step 1 and step 2 is indicated in Figure 6. The flowchart will give a more concise view of the sequnece of the instruction decoding.

OP CODE Fetch Cycle

As explained in Chapter 3, addressing registers that provide an address on the IADBUS for the OP CODE fetch cycle during the T0 or T0[OVLP] cycle are determined by the

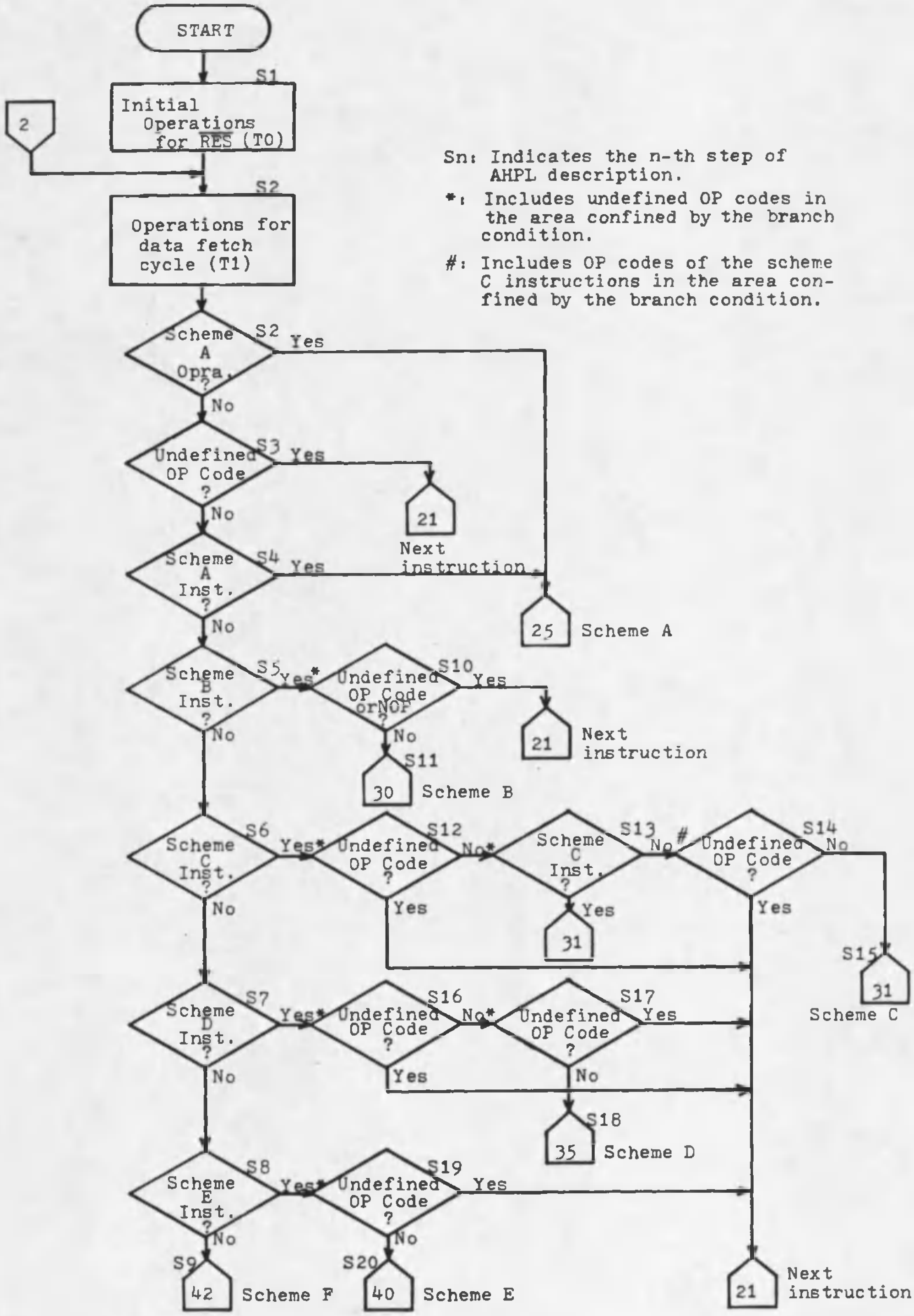


Figure 6. Flowchart for Instruction Decoding

preceding instruction. There are five combinations of the addressing registers, and they are not completely related to the addressing modes or the types of the instructions. Thus, it will be required to have a conditional connection with five lengthy conditions that select the five combinations of the addressing registers. Instead, four steps are developed to cope with the OP CODE fetch cycle. Two of the five combinations of the addressing registers are put together in one step, because they are selected by rather short conditions. Steps 21, 22, 23, and 24 will be discussed together as follows:

S21, S22, S23, and S24 -- The only difference among these steps is the use of a different combination of the addressing registers. Step 21 places a 16-bit address stored in PC on the IADBUS, while step 22 puts an address stored in INR on the IADBUS. A 16-bit address provided by DL and TR is loaded on the IADBUS in step 23. In step 24, either the catenation of INR[0:7] and ALUREG or the catenation of ALUREG and INR[8:15] produces a 16-bit address. The other operations are common to those step.

Since the address on the IADBUS during the OP CODE fetch cycle is not supposed to be incremented in the case of the internal operations of the $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ inputs. This operation will be done if either the $\overline{\text{IRQ}}$ or $\overline{\text{NMI}}$ input signal is recognized by the processor. The above condition can be

described by NMIFF + NM + (^IRQ & ^P[5]). Since the $\overline{NMI}$ input signal can be accepted in any cycle, NMIFF is used to remenber the occurence of a signal on the $\overline{NMI}$ input line, and the NM communication terminal is proviede to indicate the occurence of a signal on the $\overline{NMI}$ input line during the OP CODE fetch cycle. The $\overline{IRQ}$ input signal is acknowledged only during the OP CODE fetch cycle, if the interrupt disable flag is cleared and an $\overline{NMI}$ interrupt has not been requested.

It is assumed that an $\overline{NMI}$ interrupt has priority over an $\overline{IRQ}$ interrupt when both signals are candidates to activate their internal operations. No information about the above situation is available. However, it will be natural that the $\overline{NMI}$ input has priority. The IRQF control flip-flop to be set by the condition ^(NMIFF + NM) & ^IRQ & ^P[5] can not be used as a condition parameter for the address transfer from the IADBUS into INR. To set NMIF, the NMIFF control flip-flop must be used. NMIFF is set whenever a negative going edge on the $\overline{NMI}$ input line appears, and only cleared during the OP CODE fetch cycle. The operations to set NMIF and IRQF must be done during the OP CODE fetch cycle because the internal operation to be performed is decided during the data fetch cycle (T1). The steps associated with the OP CODE fetch cycle are shown in Figure 7.

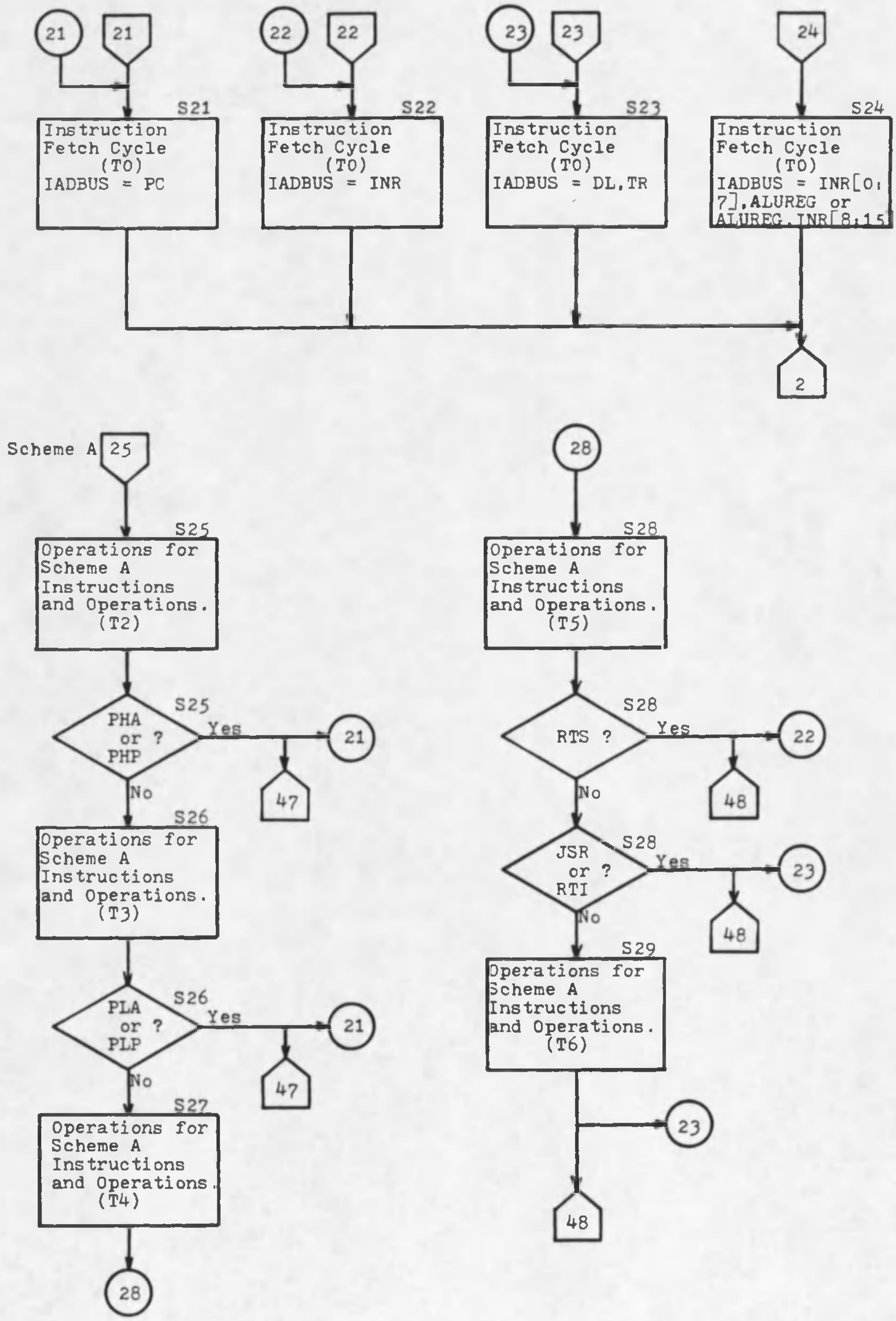


Figure 7. Flowchart for OP CODE Fetch Cycle and Scheme A

Internal Operations

The operations to be performed in the T2 cycle and the following cycles of the internal operations of all instructions and inputs must be defined after step 24 in order to complete the 6502 AHPL description. As all instructions and inputs are classified into the six schemes for efficient synthesis of all internal opeations, each scheme will be developed in turn after step 24. Following the steps for scheme F, the operations for execution parts will be defined. Since only the scheme which the OP CODE stored in IR belongs to has been determined through the steps associated with the instruction decoding, further distinction among the scheme instructions must be done by means of conditional transfers, connections, and branches.

Scheme A

The steps in association with scheme A are intended to describe the operations for the T2 cycle and the following cycles of the internal operations of the scheme A instructions and inputs. Since the BRK instruction as well as the $\overline{\text{RES}}$, $\overline{\text{IRQ}}$, and $\overline{\text{NMI}}$ inputs requires five cycles other than the T0 and T1 cycles, at least five steps should be assigned for scheme A. Step 25 through step 29 will be discussed as follows:

S25 -- Step 25 defines all operations for the T2 cycle of the scheme A instructions and inputs. In order to

synthesize the operations, related internal operation seqeunces attached in Appendix C should be studied first. The Karnaugh map in Appendix E should also be used to derive as little condition parameters as possible for conditional transfers, connections, and branches. Since the operations in step 25 are performed for the internal operations of the instructions and inputs classified into scheme A, the 8 bits of IR and the EXT, RESF, IRQF, and NMIF control flip-flops must be used as condition parameters.

The conditions having only IR bit parameters must be further conditionded with ^EXT, because IR might have stored one of the OP CODE's for the scheme A instructions in the last OP CODE fetch cycle when the processor is going to execute the internal operation of one of the inputs. The EXT is set during the T1 cycle in step 2 if requested. Instead of using the RESF + NMIF + IRQF condition, the use of the EXT parameter will shorten the conditions necessary for the operations defined in step 25 through step 29.

More than one of RESF, IRQF, and NMIF may not be set in the steps associated with scheme A in order to carry out correct conditional operations. As discussed before, IRQF and NMIF must be set only during the T0 cycle in steps 21, 22, 23, and 24. This is because that only the operations for one of the inputs must be done until the internal operation of that input has been completed. The operations

to clear IRQF and NMIF in step 1 are provided to allow the start of the internal operation of the $\overline{RES}$ input in the course of the execution of the internal operation of either the $\overline{IRQ}$ or $\overline{NMI}$ input.

The connections to and from the DBUS are conditioned by the R/W output. If the internal operations for either PHA or PHP is done in step 25, control branches to steps 21 and 47 in parallel where the next OP CODE fetch and the executional operations for either PHA or PHP will be performed, respectively. If not, control continues to step 26. Note that the internal operations of PHA and PHP require one cycle, that is step 25, other than the T0 and T1 cycles.

S26 -- Step 26 describes the operations for the T3 cycle of the internal operations of the scheme A instructions and inputs. Since the condition parameters used in the conditional operations for scheme A have already been examined, further explanations about conditions will not be necessary. Only the operations not discussed in Chapter 3 will be briefly explained here. In order to understand the meanings of the operations and the conditions, the internal operation sequences concerned with the scheme A instructions and inputs in Appendices C and D as well as the K-map in Appendix E should be referred to. The operations for PLA and PLP can be conditioned by a less

number of condition parameters because the OP CODE's for PHA and PHP have been eliminated in step 25. If the processor is executing either PLA or PLP, control jumps to steps 21 and 47, as in the case of PHA and PHP. Otherwise, control advances to the next step.

S27 -- The operations for the T4 cycle are defined in step 27. Since all the remaining instructions and inputs require more cycles to complete their internal operations, control simlpy continues to step 28. Thus, there is no branch statement in step 27.

S28 -- The operations in step 28 are described for the T5 cycle. In this step, the JSR, RTI, and RTS instructions will be branched in order to complete their executional operations and to fetch the next OP CODE. Control branches to steps 22 and 48 in parallel if the operations for RTS has been performed. The internal operation of RTS uses the OP CODE fetch cycle of step 22, and the executional operations will be done in step 48. The operations in step 48 will overlap the OP CODE fetch cycle. If the operations for either JSR or RTI has been done, control branches to steps 23 and 48. The JSR and RTI can utilize the same executional operations as RTS uses. However, the internal operations of JSR and RTI require the OP CODE fetch cycle of step 23.

S29 -- Step 29 describes the operations for the T6

cycle of the internal operations of the BRK instruction and the three inputs. Because all the operations in the T6 cycle are common to the internal operations of the remaining instructions and inputs, no conditional operation is needed. After finishing the operations, control branch to steps 23 and 48, as in the case of JSR and RTI.

A flowchart for scheme A is given in Figure 7.

Scheme B

The instructions using the accumulator or implied addressing mode are classified into scheme B. Since the scheme B instructions do not require any additional cycles for their internal operations, only one step with a NODELAY statement is required.

S30 -- Step 30 has only a conditional branch statement that describes whether the OP CODE stored in IR is for one of the instructions using the accumulator addressing mode or for one of the instructions utilizing the implied addressing mode. If it matches one of the former instructions, control branches to steps 21 and 54 for the next OP CODE fetch and its executional operations. If it matches one of the latter instructions, control branches to steps 21 and 49 for the same purpose.

A flowchart for scheme B is indicated in Figure 8.

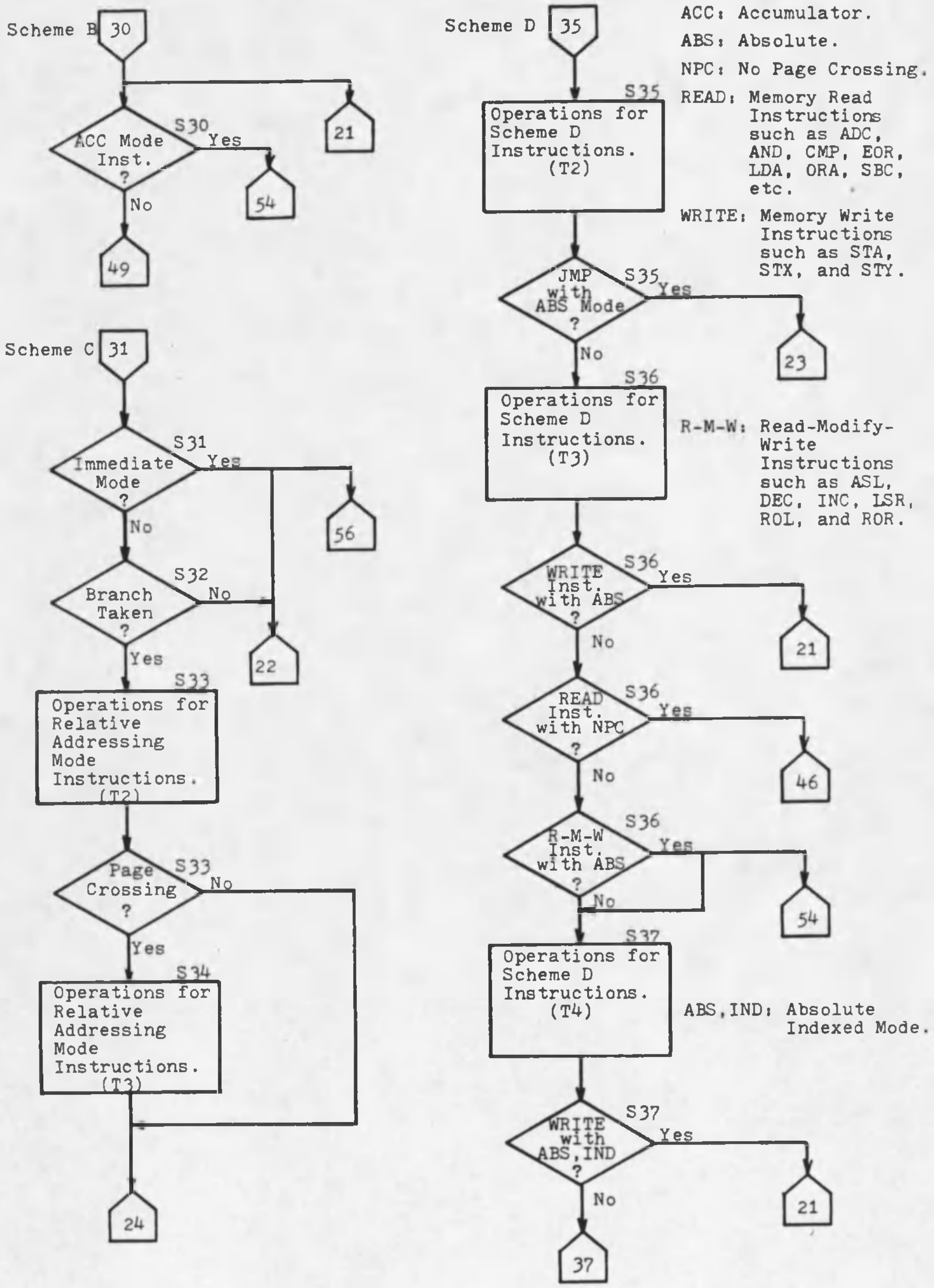


Figure 8. Flowchart for Schemes B, C, and D

Scheme C

The instructions involving the immediate or relative addressing mode are categorized into scheme C. The instructions using the immediate addressing mode do not require any additional cycles like the scheme B instructions, though the instructions utilizing the relative addressing mode must have at least two additional cycles for branch and a page crossing. In addition to two steps, another step to check branch conditions will be essential to implement the internal operations of all branch instructions. Steps 31, 32, 33, and 34 in conjunction with scheme C will be discussed as follows:

S31 -- Step 31 is assigned to select the instructions using the immediate addressing mode from the scheme C instructions. Since this step should be a NODELAY step, only a conditional branch statement with the branch condition ^IR[3] is described. If the OP CODE in IR is for one of the instructions utilizing the immediate addressing mode, control jumps to step 22 and 56 to fetch the next OP CODE and to operate its executional operations. Otherwise control continues to step 32 for the internal operation of a branch instruction.

S32 -- Step 32 is also a NODELAY step with one big conditional branch statement. Eight kinds of branch conditions are ORed and put together in the conditional

branch statement. If no branch is to be taken, control jumps to step 22 where the next OP CODE fetch will be performed by using the address in INR. Otherwise control continues to step 33 to compute a branch address.

S33 -- The operations necessary for branch address calculation are described in step 33. How to calculate a branch address has been discussed in Chapter 3. If a page crossing is not necessary, control branches to step 24 in which the next OP CODE fetch will be done. Othewise control continues to step 34 to increment or decrement the page number stored in INR[0:7]. The condition parameters of the condition to select a proper combination of the addressing registers are also prepared in step 33.

S34 -- Step 34 defines the operations for the T3 cycle of the internal operations of the branch instructions. Since the internal operations of the branch instructions have been completed in this step, control branches unconditionally to step 24 for the execution of the next instruction. No executional operation is required, as mentioned in Chapter 3.

Figure 8 shows a more apparent view of scheme C.

Scheme D

The scheme D instructions consist of the instructions using the absolute, absolute indexed, or indirect addressing mode. The number of additional cycles

the scheme D instructions use varies from one to five. Thus, at least five steps must be assigned to describe the internal operations of the scheme D instructions. Step 35 through step 39 for scheme D will be discussed as follows:

S35 -- The opeations for the T2 cycle of the internal operations of the scheme D instructions are defined in step 35. Since all of the scheme D instructions except the JMP instruction need the T3 cycle, control continues to step 36. In the case that the operations for JMP have been performed in step 35, control branches to step 23. No further operation is required for JMP. The efficient condition to select the OP CODE for JMP using the absolute addressing mode can be obtained by means of the K-map. For the understanding of the operations descrbied in step 35, the reader shall refer to the internal operation sequences involving the absolute, absolute indexed, indirect addressing mode.

S36 -- Step 36 describes the operations for the T3 cycle along with a conditional branch statement. Most of the operations are conditioned by appropriate condition parameters. It will not so hard to understand each operation in step 36 if the reader uses the internal operation sequences in Appendix C and the K-map in Appendix E. This is because that the conditional transfers, connections, and branch in step 36 have been described by

using those materials.

Since the STA, STX, and STY instructions using the absolute addressing mode terminate their internal operations in this step, control simply jumps to step 23 in the case of these instructions. In the case of the instructions such as ADC, AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, and SBC using the absolute addressing mode or the absolute indexed addressing mode without a page crossing, control branches to step 46. For the instructions such as ASL, DEC, INC, LSR, ROL, and ROR using the absolute addressing mode, control continues to step 37 and branches to step 54 for parallel operations. For the other scheme D instructions that do not satisfy the branch conditions in step 36, control continues to the next step. The condition ^C is used to indicate no page crossing.

S37 -- The operations for the T4 cycle are described in step 37. First, STA, STX, and STY using the absolute indexed mode are checked by the conditional branch statement in the step 37. If the first condition is satisfied, control branches to step 21 to execute the next OP CODE fetch cycle. No further operation is necessary. Control branches to step 46 for further operations for ADC, AND, CMP, EOR, LDA, LDX, LDY, ORA, and SBC utilizing the absolute indexed addressing mode with a page crossing. If the JMP instruction using the indirect addressing mode has been

performed, control jumps to step 23 for the next OP CODE fetch cycle. The JMP terminates its internal operation in this step. For ASL, DEC, INC, LSR, ROL, and ROR using the absolute indexed mode, control must branch to step 54 for an executional part and continue to step 38 for an addressing part. Note that only ASL, DEC, INC, LSR, ROL, and ROR utilizing the absolute addressing mode do not satisfy the conditions in the conditional branch statement. The executional opeations for these instructions are performed in parallel in step 54.

S38 -- Since the instructions that yet need additional cycles are ASL, DEC, INC, LSR, ROL, and ROR using the absolute or absolute indexed addressing mode, the operations defined in step 38 are for a write operation necessary for the these instructions. The data to be written is prepared in step 54 for absolute indexed addressing or step 55 for absolute addressing. Control jumps to step 21 for those instructions using the absolute addressing mode. Otherwise control continues to the next step for one more write operation. Since some of the scheme E instructions have the same operations in particular cycles as step 38 describes, this step will be used by those scheme E instructions.

S39 -- Step 39 describes only the operations for the T6 cycle of the internal operations of the remaining scheme

D instructions such as ASL, DEC, INC, LSR, ROL, and ROR using the absolute indexed addressing. This step will also be used for the internal operations of some of the scheme E instructions like step 38. Only a write operation is performed.

A flowchart for scheme D is provided in Figures 8 and 9 for a quick understanding.

Scheme E

The instructions using the zero page or zero page indexed addressing mode are classified into scheme E. Since some of the scheme E instructions require four additional cycles to describe their internal operations, at least four steps will be essential. However, only steps 40 and 41 are allocated for scheme E, because steps 38 and 39 for scheme D can be used, as already stated. Steps 40 and 41 in association with scheme E will be developed as follows:

S40 -- Step 40 describes the operations for the T2 cycle of the internal operations of the scheme E instructions. The conditional branch statements of step 40 and 41 will become more compact than those of the steps associated with scheme D because the scheme E instructions do not need to take a page crossing into account. Only the instructions using the zero page addressing mode are selected by the four conditions in the conditional branch statement of step 40. The STA, STX, and STY instructions

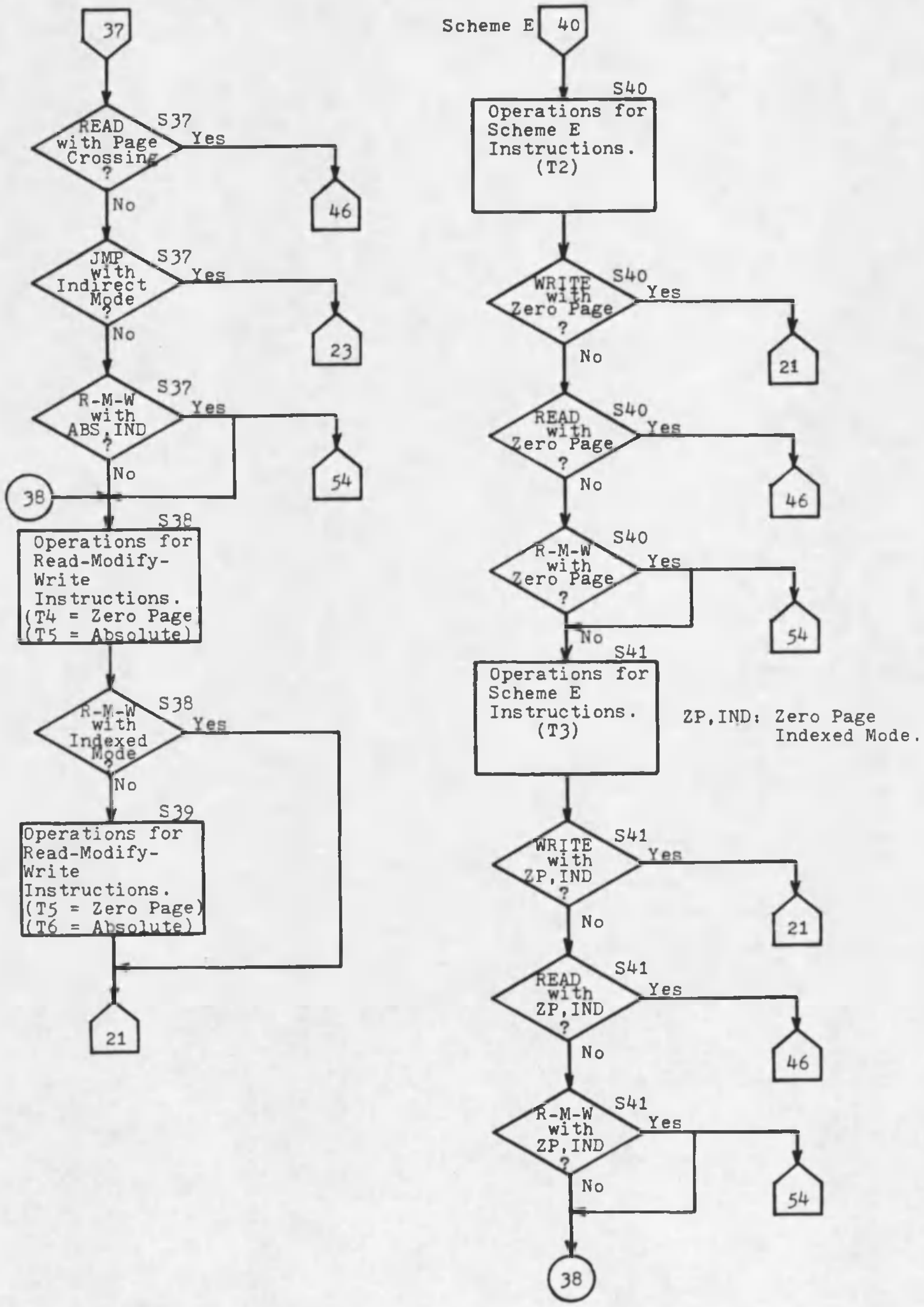


Figure 9. Flowchart for Schemes D and E

are determined by the first condition. If the condition is satisfied, control branches to step 22 for the next OP CODE fetch cycle. If the operations for one of the ADC, AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, and SBC instructions has been performed in step 40, control jumps to step 46 for the rest of the internal operation. The instructions such as ASL, DEC, INC, LSR, ROL, and ROR are determined by the third and fourth conditions for parallel operations. In this case, control branches to step 54 and continues to step 41.

S41 -- The operations for the T3 cycle of the internal operations of the instructions using the zero page indexed addressing mode and ASL, DEC, INC, LSR, ROL, and ROR utilizing the zero page addressing mode are defined in step 41. The other scheme E instructions do not require the T3 cycle. Thus, they have been eliminated in step 41.

For STA, STX, and STY, control branches to step 21 because their internal operations have been finished in step 41. Control jumps to step 46 for ADC, AND, CMP, EOR, LDA, LDX, LDY, ORA, and SBC. Since ASL, DEC, INC, LSR, ROL, and ROR empolying the zero page indexed addressing mode must start their execution parts in the next cycle, these instructions are determined by the third and fourth conditions. As the result, control branches to steps 38 and 54. These instructions use the step 38 and 39 for the T4

and T5 cycles of their internal operations. Step 41 describes a write operation for ASL, DEC, INC, LSR, ROL, and ROR using the zero page addressing mode. These instructions are also determined by the fourth condition so that control can branch to step 38 for the last cycle of their internal operations. The conditional branch statement of step 38 can also distinguish between zero page addressing and zero page indexed addressing without any modifications.

A flowchart for scheme E is indicated in Figure 9.

Scheme F

The last scheme, scheme F groups the instructions using the indexed indirect addressing mode and the instructions utilizing the indirect indexed addressing mode. At least four steps will be required to implement the internal operations of all scheme F instructions because each of these instructions use either three or four cycles other than the T0 and T1 cycles. Step 42 through step 46 will be briefly discussed in connection with scheme F.

S42 -- Step 42 describes the operations necessary for the T2 cycle of the internal operations of all scheme F instructions. No conditional branch is necessary since all the internal operations require two more cycles, namely the T3 and T4 cycles.

S43 -- All operations necessary for the T3 cycle are described in step 43. No conditional branch is required, as

in step 43.

S44 -- In addition to the operations for the T4 cycle, step 44 has a conditional branch statement to determine ADC, AND, CMP, EOR, LDA, ORA, and SBC using the indirect indexed addressing mode without a page crossing. If the operations for one of these instructions have been performed in step 44, control jumps to step 46 to complete its internal operations. No addtional cycle is necessary for those instructions. Control continues to step 45 for the other scheme F instructions.

S45 -- Step 45 defines the operations for the T5 cycle of the internal operaions of the remaining instructions. Since scheme F does not include the instructions such as ASL, DEC, INC, LSR, ROL, and ROR, only the simple condition that determines the STA instruction using the indexed indirect or indirect indexed addressing mode is inserted in the conditional branch statement of step 45. If the operations for one of the instructions have been done, control simply branches to step 21 for the next OP CODE fetch cycle. Otherwise control continues to step 46.

S46 -- Step 46 is a NODELAY step with an unconditional branch statement. Control reached this step branches to unconditionally to step 21 and 56 where the next OP CODE fetch and executional operations will be performed,

respectively. The instructions such as ADC, AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, and SBC use step 46. The purpose of this step is to provide simple conditional branch statements of the steps in which control must branch to steps 21 and 56 in parallel. This is because of avoiding writing a couple of lengthy conditions in each conditional branch statement instead of writing one condition. There are four steps where control will branch to step 46.

A flowchart for scheme F is shown in Figure 10.

Executional Operations

Most of the steps developed by now are largely related with the addressing parts of all internal operations. The execution parts of the internal operations must then be developed to complete the 6502 AHPL description. Step 47 through step 57 describes the operations necessary for the execution parts of all instructions and inputs. These steps will be discussed in brief as follows:

S47 -- Step 47 describes the executional operations for the TO[OVLP] cycle of the internal operations of the PHA, PHP, PLA, and PLP instructions. Control reached this step comes from either step 25 or step 26 defined for scheme A. The executional operations are synthesized in step 47 by means of conditional transfers and branches. Since those instructions use the stack, SP must be refreshed for the

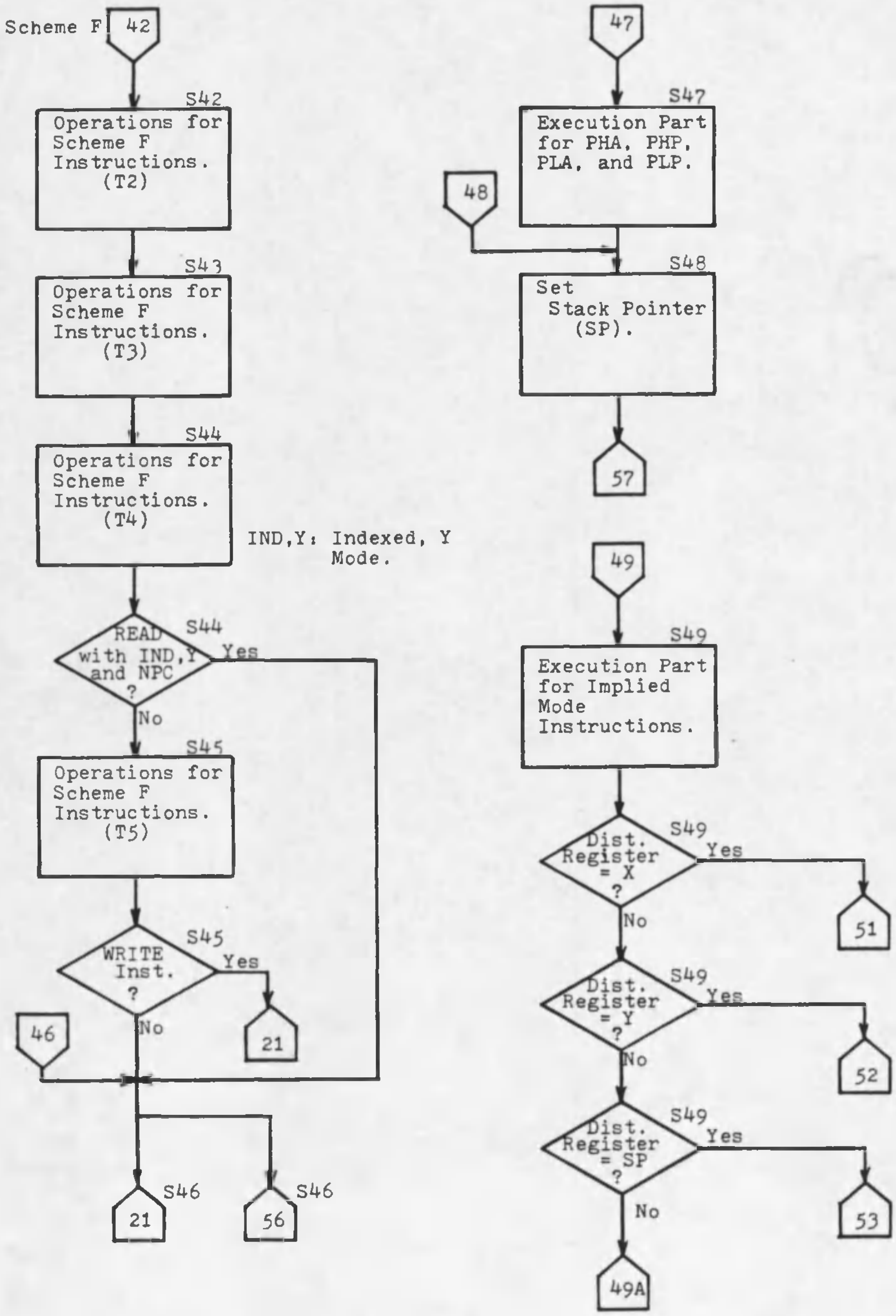


Figure 10. Flowchart for Scheme F and Executional Operations

stack-related instruction to be executed next. For this reason, control continues to the next step.

S48 -- The operations to transfer the contents of PC[8:15] into SP are described in step 48. This step is designed for all scheme A instructions and inputs because they have some operations with the stack. The single transfer operation to transfer the latest stack address stored in INR into PC has been defined in step 28 for the scheme A instructions and inputs except the instructions using step 47. After finishing the operations, control branches to step 57 to terminate itself. Note that all executional operations in step 47 through step 57 are performed in parallel while either the OP CODE fetch or the following data fetch is being executed.

S49 -- The executional operations for the TO[OVLP] cycle of the internal operations of the scheme B instructions using the implied addressing mode are defined in step 49. Main operations in this step are to refresh the P register and to execute an arithmetic operation. Since the 8-bit resultant data in ALUREG must be transferred into one of the destination registers AC, X, Y, and SP, control must be properly branched to a step to describe operations to transfer data stored in ALUREG into a destination registers.

First, by the first condition, in the conditional

branch statement, the set/clear instructions are determined. In this case, control branches to step 57. The instructions whose destination register is X are determined by the second condition. If the condition is satisfied, control branches to step 51. The instructions whose destination register is Y and the TXS instrucction are decided by the third and fourth conditions, respectively. Control branches to step 52 for the former instructions. For the latter instruction, control jumps to step 53. The instructions not selected by these conditions are using AC as a destination register. Control continues to the next step in this case. Note that those transfer operations can not be put together in one step because the contents of IR will be changed by the OP CODE fetch cycle which is performed in parallel. In other words, the steps for data transfers can not use IR as a condition parameter. Since the NOP instruction has been removed at the stage of the instruction decoding, the conditions in step 49 are using a less number of condition parameters.

S50 -- Step 50 describes the operations to transfer data from ALUREG into AC.

S51 -- Data stored in ALUREG is transfered into X by step 51.

S52 -- Step 52 moves data stored in ALUREG into the Y regsiter.

S53 -- The operations to transfer data stored in ALUREG into SP are described in step 53.

S54 -- Step 54 describes the executional opeartions for the instructions such as ASL, DEC, INC, LSR, ROL, and ROR. Six types of logic operations are performed. All addressing modes used by these instructions except the accumulator addressing mode require the same operations in this step. Therefore, only the accumulator addressing mode must be distinguished from the others. Instead of modifying data in memory, the instructions using the accumulator addressing mode modify the contents of AC.

In the conditional branch statement, these instructions are determined by the condition ^IR[5]. If the condition is satisfied, control branches to step 50 where the resultant data stored in ALUREG will be transferred into AC. Otherwise control continues to the next step. Step 50 has already been developed for the executional operations for the scheme B instructions using the implied addressing mode.

S55 -- The resultant data stored in ALUREG which is to be stored in a memory space is placed on the IDBUS by the operation described in step 55. The other operations necessary for a write operation are performed in parallel. Control simply branches to step 57 because no more operation is required.

S56 -- Step 56 defines the executional operations such as arithmetic operations and set/clear operations for the P register. The instructions using step 56 for their executional parts are ADC, AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, and SBC. One of the three destination registers, AC, X, and Y must be selected depending on the instruction currently performed by the processor. As mentioned before, no further distinction among these instructions can not be done because the contents of IR will be changed in the next cycle.

By the first condition, the instructions such as ADC, AND, CMP, EOR, LDA, ORA, and SBC are determined. These instruction are using AC as a destination register. If the condition is true, control branches to step 50 for a data transfer into AC. The CPX and LDX instructions are selected by the second instructions. If it is satisfied, control branches to step 51. The CPY and LDY instructions are determined by the third condition. Control branches to step 52 for these instructions. The instruction not chosen by the three conditions is BIT. The BIT does not require a transfer operation. In this case, control continues to the last step where control will be terminated.

S57 -- Step 57 is a DEADEND step which terminates control arrived in the step.

A flowchart for the executional operations is

clearely displayeo in Figures 10 and 11.

End of Sequence

By this point, Step 1 through step 57 of a control sequence for the 6502 have been developed. However, in addition to these steps, several operations must be described after the end of the sequence to complete the 6502 AHPL description. The operations described after the end of the sequence are performed every cycle regardless of the position of control.

First, the $\overline{RES}$ and $\overline{NMI}$ inputs for which the processer must take action immediately are checked every cycle. If a negative going edge occurs on the $\overline{RES}$ input, the RESF control is set. This operation can also be implemented by placing RESF <= \1\ in step 1 in a control sequence to be run on the new compiler. However, the operation is intentionally placed after the end of the sequence in conjunction with a control sequence designed for simulation. The control flip-flops, FF1 and FF2 are used to cope with the $\overline{NMI}$ input that is an edge sensitive input. The NMIFF and NM used in steps 21, 22, 23, and 24 are set by the ^FF2 & ^NMI condition. The overflow flag, P[1] is also set here by a negative going edge on the S.O. input during the phase 1 clock pulse.

Since the ADBUS is always connected with the IADBUS, the connection is defined here. The OSC communication

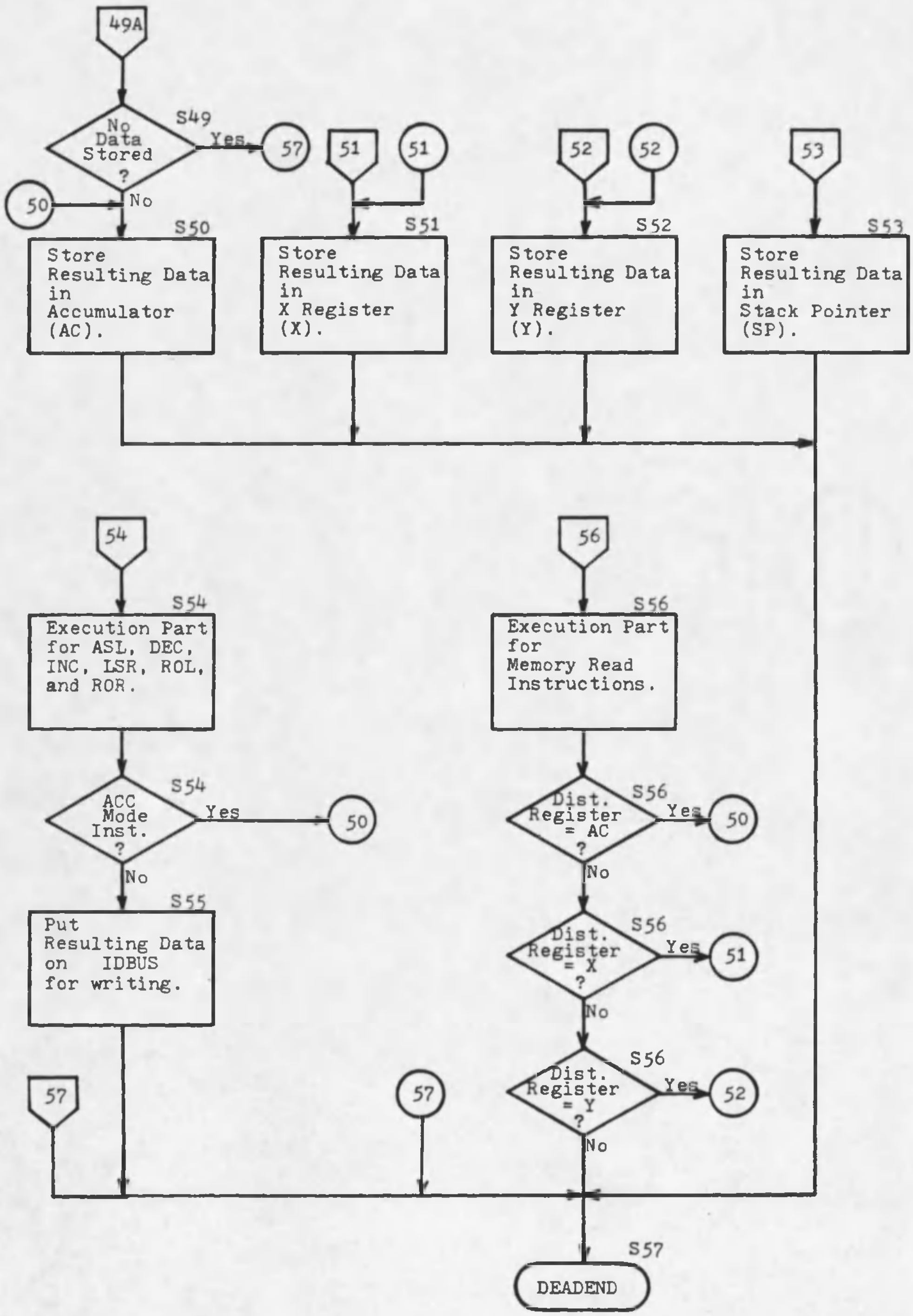


Figure 11. Flowchart for Executional Operations

terminal implies the use of the output of a clock generator. However, OSC is merely connected to the PHIO input because there is no way to express analog circuits in AHPL. The communication terminal, CLOCK is conditioned by PHI2 and ONE, which is in turn conditioned by the RDY input. The CLOCK is used as a clock signal connected to the clock inputs of all control D flip-flops.

Finally, the operation to transfer the resultant data from the ALU into ALUREG is described according to one of the hypotheses which is assuming that ALUREG can not store data for more than one cycle. Since the decimal and carry flags are always connected to the PS argument, the connection is described here instead of writing the connection in those steps in which an arithmetic or logic operation will be performed.

Design of Combinational Logic Units

As discussed in the section, Declarations, the ALU, INC, and DEC units are declared in the 6502 AHPL description. These CLU's must be individually designed to support the AHPL description. The syntax and semantics of the expressions used in designing a CLU are extensively described in reference 7. The design of each will be explained in brief.

ALU

The ALU declared in the AHPL description is a complicated combinational logic unit because it has as many as 11 kinds of arithmetic and logic operations. The specification of the ALU has been shown in Table 4 and has been discussed in Chapter 3. The ALU consists of a decimal adder named DADD and a binary adder named BADD. The DADD is used for the arithmetic operations specified by operation codes 1 and 5. Decimal addition must be performed for the ADC and SBC instructions if the PS[0] = 0. The contents of argument 2 are complemented in the ALU for subtraction.

To get the results of an arithmetic or logic operation, the ALU executes all arithmetic and logic operations prepared in the ALU with the same four arguments. However, the ALU outputs only the result of the operation specified by the current operation code. Descriptions of ALUNIT, ADDER, DECADDER, and FULLADDER necessary for implementing the ALU have been given in Appendix G. Since the design of ADDER and FULLADDER is simple enough, no explanation will be required, though an explanation for the design of DECADDER must be presented. It has been left to the next chapter because a 6502 AHPL description developed for simulation describes operations to carry out decimal addition and subtraction.

INC

The INC unit used in the AHPL description is desribed in the INCR unit. The INCR description in Appendix G will explain the function of INC precisely.

DEC

The DEC unit has been described by the name of DECR. The only difference from INCR is the use of the complement of X[J÷1:15]. A DECR description has also been shown in Appendix G.

CHAPTER 5

SIMULATION

A 6502 AHPL description has been completely developed by means of the internal operation sequences given in Appendices C and D. Now, it is ready to translate it into an SLA form for a VLSI implemetation. However, there is no evidence that the internal operations of the 6502 are correctly described in the AHPL description.

As stated in Chapter 1, compiler programs have been introduced to assist designers in designing or developing digital systems. One of such programs called HPSIM2 has been implemented in order to debug the digital systems written in AHPL. It is a matter of course in the area of VLSI design that a circuit should be completely checked before the VLSI implementation of the circuit. According to this rule, the 6502 AHPL description should be verified by using the HPSIM2 simulator. In this chapter, the HPSIM2 simulator, modifications for simulation, and the results of the simulation will be briefly discussed.

HPSIM2

The 6502 AHPL description developed through Chapter 4 is to be run on the new multi-stage compiler for

translation into the form of an SLA. Before proceeding to the translation, the verification of the AHPL description by means of a simulator is expected. Unfortunately, a simulator based on the new compiler was not avilable at the time of the development of the AHPL description. The only simulator available was HPSIM2.

The HPSIM2 can accept only the descriptions written in the exsisting AHPL but not the AHPL descriptions written in the universal AHPL. For example, HPSIM2 can not accept tailored CLU's such as ALU, INC, and DEC described in Appendix G. The HPSIM2 has some standard CLU's essential for the design of digital systems in itself instead. The procedures for preparing an AHPL description for HPSIM2 has been described in reference 6.

Modifications for Simulation

In order to make use of HPSIM2, the 6502 AHPL description must be changed into a description that HPSIM2 can accept. As mentioned in the preceding section, HPSIM2 can not accept tailored CLU's. The use of the standard CLU's provided in HPSIM2 in place of the three CLU's declared in the 6502 AHPL description is a major modification.

The Use of Five Adders for Arithmetic Operations -- Five adders are declared in a CLUNITS section to compensate for all kinds of arithmetic operations prepared in the ALU.

The ADD2, ADD3, ADD4, and ADD5 are used only for decimal addition and subtraction. For subtraction, the contents of the argument to subtract are complemented outside an adder. The ADD1 is used for address calculation and 8-bit binary addition. The ADD1, ADD2, and ADD3 can adds the carry input, while ADD4 and ADD5 do not have the carry input. Several communication terminals are declared in an OUTPUTS section to cope with decimal addition and subtraction.

To operate decimal addition and subtraction, the low order 4 bits of data on the IDBUS and the low order 4 bits of data on the ARGBUS are added together with the carry input in ADD2. Then, the most significant bit of the resultant 5-bit data from ADD2 is connected to the CD1 communication terminal and the other 4 bits are connected to the LOW communication terminal. For addition, if the resultant data is less than 10, the 4 bits connected to LOW are connected to the DECOUT[5:8] communication terminal, and a binary zero is connected to CD2 to express that the carry output is 0. Otherwise, the 4 bits are added to CONST which has a value of 6 in ADD4. The 4 bits of the resulatant 5-bit data from ADD4 exclusive of the most significant bit is connected to DECOUT[5:8]. The most siginificant bit is merely discarded. Instead, a binary one is connected to CD2 to indicate that the carry output is 1.

For subtraction, if the resultant 5-bit data

connected to CD1 and LOW is greater than 16, the 5-bit data is directly connected to CD2 and DECOUT[5:8]. Otherwise the 4 bits connected to LOW is added to CONST which has a value of 10 in ADD4. The most significant bit connected to CD1 is connected to CD2. The lower 4 bits from ADD4 are connected to DECOUT[5:8]. By one of the above operations, the values of CD2 and DECOUT[5:8] are computed. The DECOUT[5:8] has now the low order 4 bits of the result of decimal calculation. The CD2 is used as the carry input for the calculation of high order 4 bits.

Once the low order 4 bits have been obtained, the high order 4 bits on the ARGBUS and the high order 4 bits on the IDBUS are added together with CD2 in ADD3. The resultant 5-bit data from ADD3 is connected to CD3 and UP. If the resultant data is less than 10, the 4 bits connected to UP is connected to the DECOUT[1:4], and a binary zero is connected to DECOUT[0] for decimal addition. Othewise the contents of UP is added to CONST = 6 in ADD5. A binary one is connected to DECOUT[0], and the lower 4 bits of the resultant 5-bit data from ADD5 is connected to DECOUT[1:4].

For decimal subtraction, the resultant 5-bit data connected to CD3 and UP is directly connected to DECOUT[0:5] if the resultant data is greater than 16. If not, the 4 bits connected to UP is added to CONST = 10 in ADD5, and then CD3 and ADD5[1:4] are connected to DECOUT[0:4]. 8-bit

decimal addition and subtraction are performed in this way. The DECOUT[0:8] has now the result of decimal calculation. Those operations for decimal addition and subtraction can be seen in step 59 of the 6502 AHPL description attached in Appendix H, which demonstrates an example of simulation.

Description for Logic Operations -- Since there is no standard CLU having a function such as AND, OR, exclusive OR, shift, or rotation, each function for a logic operation must be individually described in AHPL in the step concerned with the logic operation. Each logic operation will be selected by an appropriate condition.

Control of Data Transfers into ALUREG -- To control data transfers into ALUREG, the control flip-flop, INH is declared in a control sequence aimed at simulation. The INH will inhibit the resultant data from ADD1 from being transferred into ALUREG when the data generated by an arithmetic or logic operation other than the arithmetic operation using ADD1 must be stored into ALUREG. The 8-bit binary addition in ADD1 is performed every cycle since the operation is described after the end of the sequence according to one of the hypotheses discusssed in Chapter 3.

An internal operation to require an arithmetic or logic operation other than the arithmetic operation using ADD1 must use INH. The INH is set in the cycle immediately before the cycle in which the arithmetic or logic operation

is performed, and INH is cleared as soon as the operation has been done. It is not expected to set INH as soon as an internal operation has been determined in the T1 cycle because the internal operation might use the 8-bit binary addition using ADD1 for its effective address calculation.

RES Condition Parameter -- The HPSIM2 does not have the capability that the CONTROLRESET(1) statement is activated by a designated input, unlike the new multi-stage compiler. Thus, the conditions in the branch statements of all non-NODELAY steps must be conditioned by the $\overline{\text{RES}}$ input so that control can branch immediatelyto step 1 where the T0 cycle of the internal operation of $\overline{\text{RES}}$ will be performed, when a negative going edge on the $\overline{\text{RES}}$ input line appears.

Memory Modules -- To describe memory modules after a 6502 AHPL description used for simulation is not a direct modification of the 6502 AHPL description to be run on the new compiler. It is less complicated to write OP CODE's and data in memory modules than to place proper input data directly on input lines in order to simulate the executions of instructions.

Four memory modules, RAM1, RAM2, RAM3, and ROM have been described. The RAM1 defines a 256 8-bit memory module placed in page 0. The 256 locations of RAM1 are provided for the instructions using the zero page or zero page indexed addressing mode. The RAM2 has the same memory

spaces as RAM1 has, but is placed in page 1 to be used as the stack. 256 locations in page 2 are provided by RAM3 as program memory. The 1024 8-bit read-only memory, ROM covers pages 252, 253, 254, 255 to store starting addresses of interrupt routines. The reader should remind that the vector locations FFFA, FFFB, FFFC, FFFD, FFFE, and FFFF are addressed by the internal operations of the $\overline{\text{RES}}$, $\overline{\text{IRQ}}$, and $\overline{\text{NMI}}$ inputs. A modified 6502 AHPL description has been given in Appendix H.

Results

Once a description including the memory modules has been completed, the internal operations of the 6502 will be verified by using HPSIM2. OP CODE's, program data, and input signals are assigned through an HPSIM communication section. The internal operations of all instructions and inputs can be confirmed by defining them in several communication sections. Although the instructions categorized into the same fundamental instructions such as the branch and transfer instructions have some common operations, it is desirable to check that all conditional transfers, connections, and braches for each internal operation are properly performed.

Since all internal operations have been verified, as the last step of simulation, an application program should be run on the 6502 simulated by HPSIM2 in order to

demonstrate the completeness of the 6502 AHPL description developed for the simulation. An 8-bit multiplication program is selected for that purpose. For easy understanding of how the program is to multiply two 8-bit binary numbers, a flowchart of 8-bit multiplication is indicated in Figure 12. In addition, the 8-bit multiplication program to be run has been given in Figure 13 in the assembly language form with enough comments in order to help to comprehend the meanings of the OP CODE's, program data, and input signals placed in the HPSIM communication section attached in Appendix H. These two figures are cited from reference 5.

The program requires location 0200 through 0222 in hexadecimal. Since RAM3 is used for page 2, the program is actually written in M3<0:34> in the communication section. A multiplicand and multiplier are stored in locations 0000 and 0002, respectively. So, an FF for a multiplicand and an FF for a multiplier are written in M1<0> and M1<2> of the RAM1 module. Because the program is to be intiated by the RES input, location 0200 must be previously set in the vector locations, FFFC and FFFD. Therefore, a 02 and a 00 are stored in M4<1021> and M4<1020> of the ROM module, respectively.

As a result of the simulation, M1<3:4> has a 01FE in the list of the dump of the memories. The result is correct

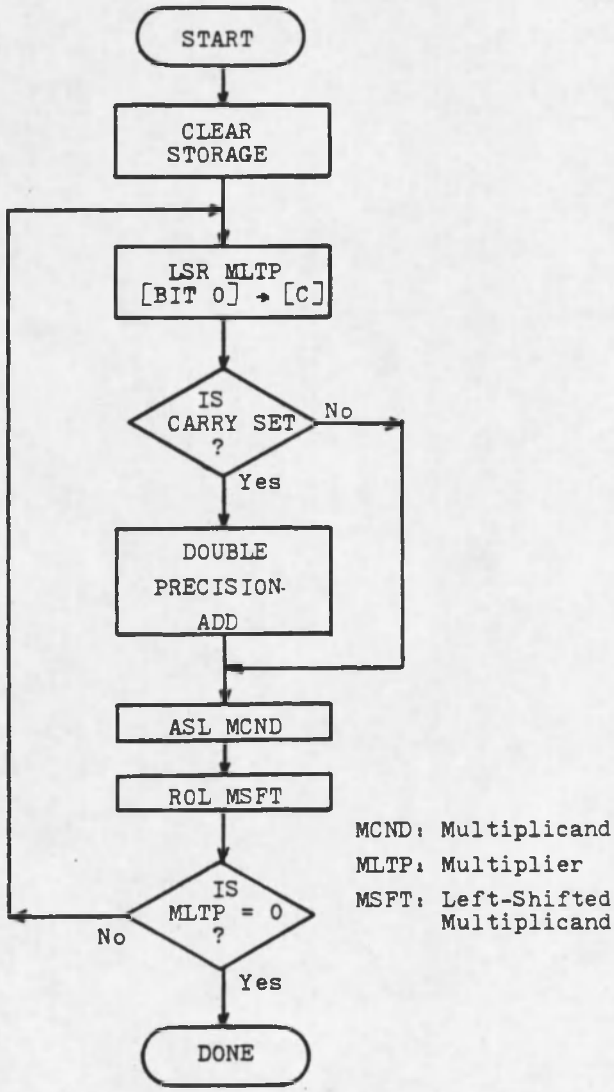


Figure 12. Flowchart of 8-bit Multiplication

```
$0000 = MCND; Multiplicand.
$0001 = MSFT; Multiplicand is shifted into this location.
$0002 = MLTP; Multiplier.
$0003 = PRDLO; Low-order byte of the product.
$0004 = PRDHI; High-order byte of the product.


0200  D8      START   CLD         Clear decimal mode.
0201  A9 00           LDA $00     Clear Storage locations for
0203  85 01           STA MSFT    MSFT, PRDLO, and PRDHI.
0205  85 03           STA PRDLO
0207  85 04           STA PRDHI
0209  46 02   AGAIN   LSR MLTP    Shift multiplier right into
020B  90 0D           BCC ARND    carry flag to test for one
                                  or zero.
020D  18              CLC         Clear carry flag for
                                  addition.
020E  A5 00           LDA MCND    Get multiplicand.
0210  65 03           ADC PRDLO   Add to low-order byte
                                  of product.
0212  83 03           STA PRDLO   Store result.
0214  A5 01           LDA MSFT    Get shifted multiplicand.
0216  65 04           ADC PRDHI   Add to high-order byte
                                  of product.
0218  85 04           STA PRDHI   Store result.
021A  06 00   ARND    ASL MCND    Shift multiplicand and
021C  26 01           ROL MSFT    roll it into MSFT
                                     (Multiplicand shifted).
021E  A5 02           LDA MLTP    If multiplier is not zero
0220  D0 E7           BNE AGAIN   then the job is not
                                  again.
0222  00      DONE    BRK         Otherwise job is finished.
```

Figure 13. 8-bit Multiplication Program

because FF x FF = FE01. It took 370 clocks to compute the program. The results of the simulation have been shown in Appendix H. The reader should note that the control sequence used for the simulation does not have the same number of steps the AHPL description in Appendix F uses, and that the instructions in the program are using only a few kinds of addressing modes. The AHPL description developed for an SLA realization is written more effectively. Some mistakes have been found since the last simulation listed in Appendix H was done.

CHAPTER 6

SLA REALIZATION

By now, a 6502 AHPL description to be run on the new compiler has been completed for an SLA realization. Furthermore, the internal operations of the 6502 AHPL description have been verified on HPSIM2 by using a modified 6502 AHPL description developed for simulation. The remainder of the project is to translate from the 6502 AHPL description into a VLSI implementation in the form of an SLA.

However, only fundamental prerequisites for an SLA realization will be discussed here in order to prepare for the translation. In fact, the translation will not be done through this paper. Chapter 6 will present a brief introduction to an SLA, an overview of an algorithm for an SLA realization, and the discussion on expected outputs of the SLA realization.

Introduction to a Storage Logic Array

A storage logic array (SLA) described in reference 2 was invented from the concept of a programmable logic array (PLA) clearly mentioned in reference 8. The SLA is composed of flip-flops and the PLA portion that contains an AND array

and an OR array to produce multiple Boolean outputs from its inputs. In addition to the use of flip-flops, the SLA differs from the PLA in that the AND array and OR array are folded together. For this characteristic, the rows of the array can be further divided into multiple independent segments, and more flip-flops can be added in each segment. The columns of the array can also be subdivided by extending these concepts, so that more flip-flops can be placed along the columns of the array. The concepts of row segmentation and column segmentation as well as the use of flip-flops for local feedback adopted in the SLA will improve array utilization efficiency.

Since the primary aim of the use of the SLA is to provide the direct translation from AHPL descriptions into VLSI implementations by a computer, it is more desirable to use clocked D flip-flops as the storage elements than to employ RS flip-flops in the original SLA. D flip-flops are usually used for the hardware realizations of AHPL descriptions. The clocked SLA including clocked D flip-flops must connect three additional column wires, that is, clock, +5 volts, and ground to each clocked D flip-flop. By these connections, a clock signal need not be programmed explicitly in the clocked SLA unlike the original SLA. A pictorial representation of the clocked SLA is shown in Figure 14 for the easy of understanding.

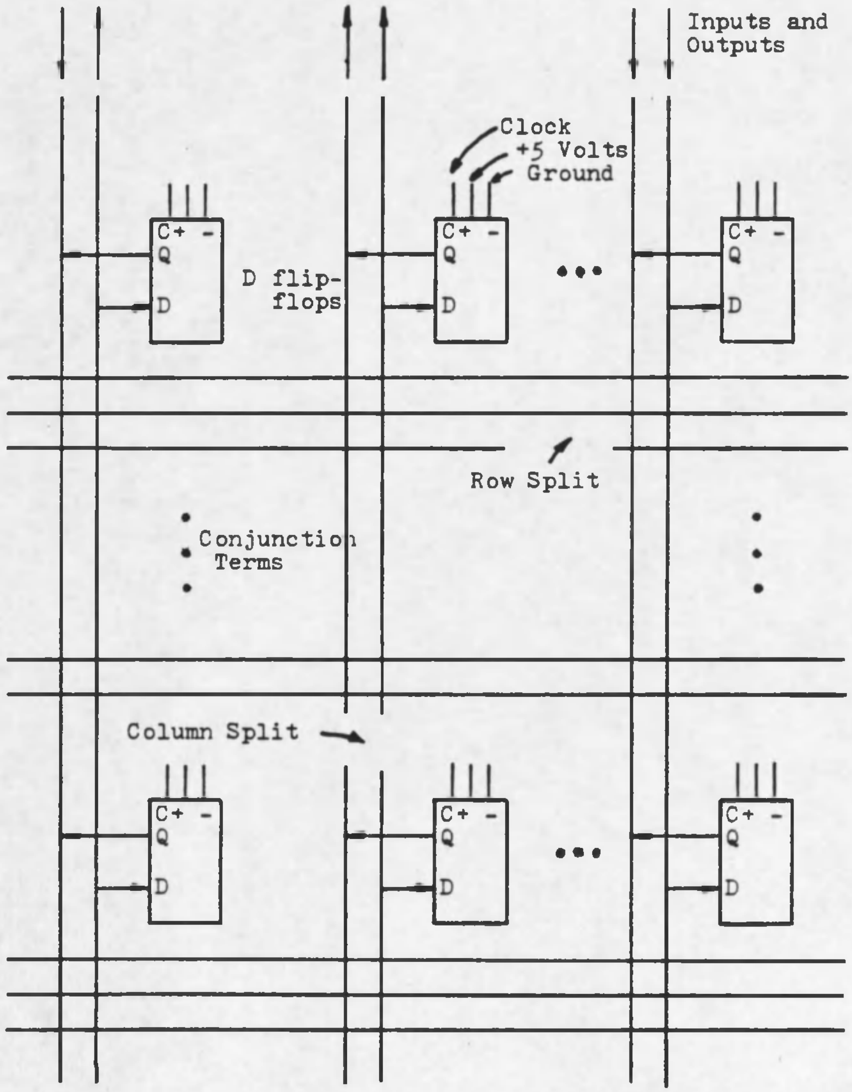


Figure 14. Clocked Storage Logic Array

It is apparent that the detailed transistor representation for the PLA portion of the SLA will not be essential to express connections between rows and columns. To accomplish a concise illustration, symbolic SLA format has been introduced. The reader should refer to reference 9 for further information regarding the clocked SLA and symbolic SLA format, because the clocked SLA will be utilized exclusively for VLSI implementations of the digital systems described in the universal AHPL.

Algorithm for an SLA Realization

In this section, an algorithm for translating an AHPL description into an SLA layout will be briefly described in order to give the reader the background for further discussion of translation procedures. The reader is expected to refer to reference 10, which describes the algorithm in detail, before proceeding to actual translation.

As already stated, the new compiler consists of multiple stages. A compiler to be used for an SLA realization will have three stages. These stages will be explained in connection with the algorithm for an SLA realization. Stage 1 and stage 2 of the new multi-stage compiler are implemented for general purpose. Naturally, the compiler capable of translating an AHPL description into an SLA layout will take advantage of these two stages. The

only task necessary for implementing the compiler is to define a method of the translation, that is, to create a stage 3 algorithm.

As the first step, stage 1 checks the syntax of all expressions defined in an AHPL description. As a result, stage 1 generates appropriate executable tables. From the executable tables, stage 2 will produce segment information such as a segment table and a segment information table. First, stage 2 finds segments in a description, and then determines the types of the segments. There are 10 types of segments regulated in the algorithm, what is called, stage 3 row ordering algorithm. Only one segment table that may contain several modules will be produced. The definitions of the 10 types of segments should be referred to in reference 10. A segment table consists of only three columns that include segment-id numbers, segments, and the types of the segments.

As the next step, stage 2 produces a segment information table by adding source and control information to each segment. Depending on the type of a segment, sources of the segment, the control logic to control the sources, and clock condition are included in the segment information table. In the case of CLU segments, the bit by bit logic is also placed in the table. Those tables are directly avilable from segmnet-oriented stage 2.

Following the processes of stage 2, stage 3 must take care of the rest of the translation task. Stage 3 is divided into stage 3A and stage 3B aimed at segment processing and bit by bit logic generation, respectively. From the segment information produced through stage 2, stage 3A can now generate connectivity information for the segments identified in the segment table. First, stage 3A produces a source/destination table that contains a source list and a destination list of each segment. New segments will be appended in the source/destination table as the processes of stage 3A proceed. The main processes on the table are to find horizontal segments that can be connected by vertical segments without using any additional columns, to produce all necessary vertical segments to connect horizontal segments, and to create all necessary horizontal segments to connect vertical segments. The categorization of segment is explained in reference 10.

After those processes, additional segments may be added at the end of the original source/destination table if some horizontal segments require vertical segments for connecting them or some vertical segments need horizontal segments for connecting them. Three types of such segments are defined to distinguish between original segments and added segments. The manner to find connections of horizontal segments and vertical segments that result in the

use of the least number of additional segments has been discussed in reference 10. As a result of those processes, an extended source/destination table that includes connectivity information is completed.

Stage 3A continues to produce adjacency information such as an adjacency table. The adjacency information can be obtained from the connectivity information generated at the previous step. First, a connection table that describes all the segment connections together with the connections of a superficial horizontal segment used for outside world communication is produced to help the efficient merging of vertical segments.

By using the connection table, partitions on vertical segments will be performed. Once the partition with the fewest classes of vertical segments has been found, a specific ordering of horizontal segments must be examined. Several relation tables for that partition may be checked to find a valid relation. As the final step of stage 3A, the adjacency information in the form of the adjacency array table that specifies the merging of vertical segments, ordering of horizontal segments, and connections of the vertical and horizontal segments is produced. The ordering of the horizontal segments can be obtained from a valid relation table. Detailed information about the merging and ordering techniques is also described in reference 10.

Finally, stage 3B determines the positions of each bit of segments in the extended source/destination table. The method of assigning bit positions of specifies a final SLA layout. An output flow of SLA translation is indicated in Figure 15 for the easy of understanding.

Expected Outputs of SLA Translation

It should be obvious that several outputs must be produced through SLA translation, since an overview of the algorithm for an SLA realization has already been presented in the preceding section. The first output of the SLA translation will be a segment table, and the next output is a segment information table. These two tables are produced through stage 2. As the reader has already noticed, the 6502 AHPL description in Appendix F is so big that it will be laborious to produce these tables manually. It is hoped that manual translation can be started with an original source/destination table. This will be justified by the fact that only mechanical act will be required to produce the segment table, segment information table, and original source/destination table.

Anyway, these tables are essential outputs for later processes of the SLA translation. Once the original source/destination table has been obtained, additional vertical segments and horizontal segments must be determined intellectually according to the rule described in reference

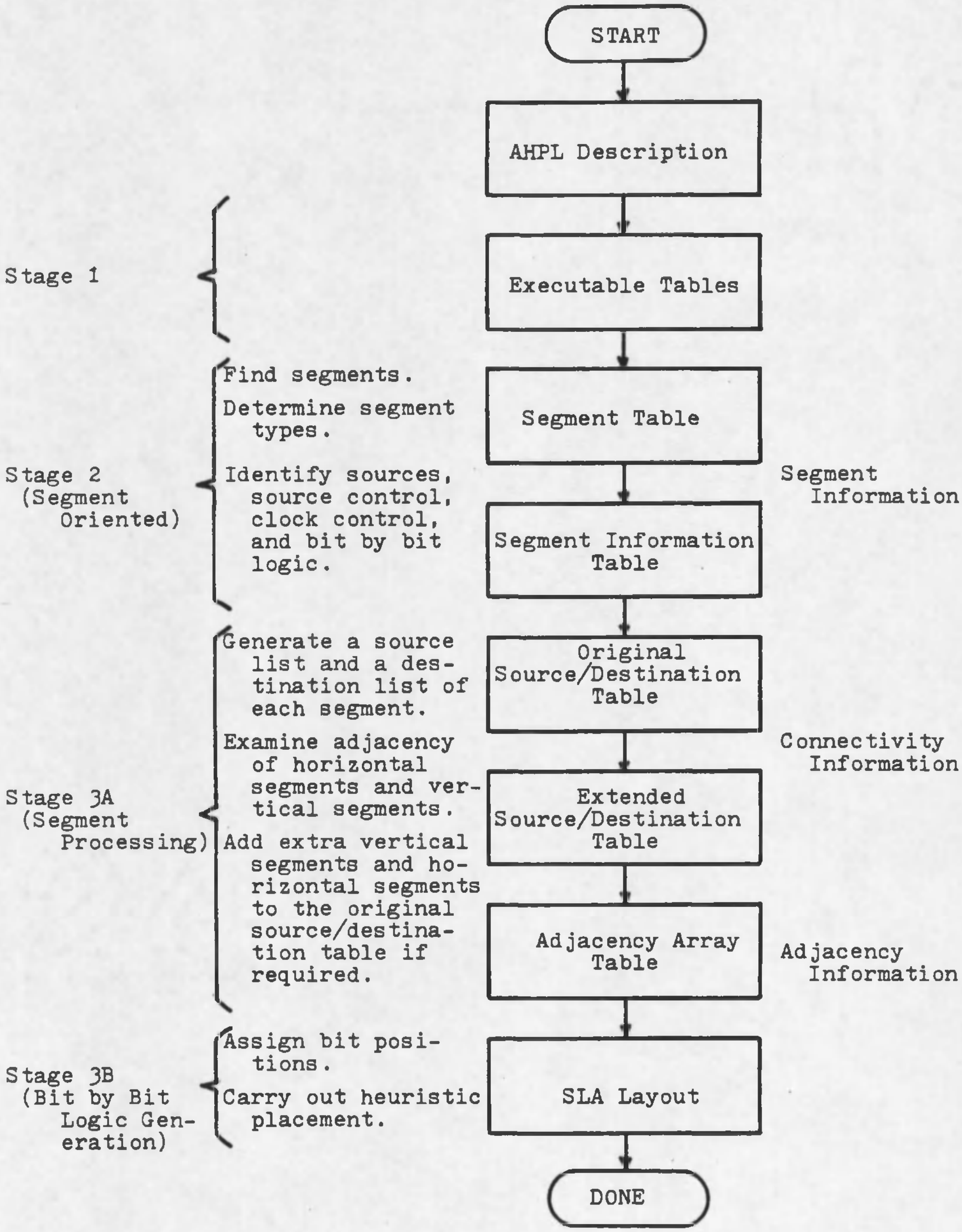


Figure 15. Output Flow of SLA Translation

10, and then added to the source/destination table to constitute an extended source/destination table. The results of this task will be fedback to the design of the algorithm in order to realize an SLA with higher density.

Like the extended source/destination table, an adjacency array is another important output of the SLA translation. It will also affect the design of the algorithm considerably. The next task to assign bit positions of each segment will be performed mechanically, as mentioned in reference 10. Finally, the last output, an SLA layout will be produced by utilizing the adjacency array table and bit positions assigned in the connectivity list.

CHAPTER 7

DISCUSSION AND CONCLUSION

Through this paper, analysis of the internal operations of the 6502 microprocessor, the development of an 6502 AHPL description, and the verification of the description have been discussed with emphasis, while translation from the 6502 AHPL description into a VLSI implementation in the form of an SLA has been described in brief. Thus, it is natural that the discussion in this chapter will be centered on the first three tasks.

With respect to the 6502 internal architecture adopted for internal operations analysis, some problems have occurred. Two of them deserve to be discussed here.

First, TR has been provided in the internal architecture to store data temporarily, though the actual hardware of the 6502 uses ALUREG as a temporary register. In fact, the internal operations of all instructions and inputs were analyzed in accordance with the actual hardware, provided that the ARGBUS and IDBUS can be flexiably connected to either argument 1 or argument 2. However, it was proved that additional buses to interchange those two buses would be necessary, after the first version of

internal operation sequeces had been completed. So, the internal operation sequences were examined, provided that the ARGBUS must always be connected to argument 1 and the IDBUS must always be connected to argument 2. As a result, unavoidable conflict had been found in the internal operation sequences of the JSR and some other instructions. The only way in which the conflict can be solved was to use TR in lieu of ALUREG. This is why TR has been provided in the internal architecture.

Second, in the load instruction, data to be stored is routed to the ALU before stored in a destination register. It is possible that the P register could be refreshed when the data is on the IDBUS, so that no routing to the ALU will be necessary. This modification requires small additional hardware. However, it is author's design policy that the uniformity of the operations as long as the internal operations meet the specification of the 6502.

As for the 6502 AHPL description developed through this paper, some discussion should be given before conclusion. The AHPL description has been written in the manner that as large block of operations as possible should be placed in a single step.

Steps 21, 22, 23, and 24 describing the OP CODE fetch cycle may be put together in a single step. In the course of the development of the AHPL description, no

relation between the addressing registers and the instructions was able to be seen. After several modifications of the internal operation sequences, the addressing registers used in the T0[OVLP] cycle of the sequences were finally determined, as indicated in Table 3. In fact, the use of PC as an addressing register had been intentionally considered in each sequence. Now, some relations will be derived so that all combinations of the addressing registers can be selected by sufficiently short conditions in a single step.

Since the 6502 AHPL description has been developed by synthesizing operations described in the internal operation sequences, it tends to have unnecessary conditional transfers and connections for operations which require simple unconditional transfers and connections. This can be reduced by a comprehensive trace of the internal operations described in the AHPL description.

The actual hardware of the 6502 also uses a timing controller to control all operations that take place in one of the T0, T1, T2, T3, T4, T5, and T6 cycles. In the AHPL description, the use of the timing controller implies that all operations to be performed in one of those cycles must be described within one step. As stated in Chapter 4, it will cause the step to have lengthy conditions of conditional operations.

In conclusion, it is apparent that at least 147 D flip-flops will by required in an SLA, though the 6502 AHPL description has not yet been translated into an SLA layout. The SLA layout will have to take a clock generator not described in the AHPL description into condideration. A completely verified 6502 AHPL description with a moderate number of steps has been developed, and a study of translation procedures has been done as part of the SLA compilation project.

APPENDIX A

6502 INSTRUCTION SET SUMMARY

Instructions Mnemonic	Operation	Addressing Mode	OP	N	#	N	V	B	D	I	Z	C
ADC	A + M + C -> A (1)	IMMEDIATE	69	2	2	N	V	•	•	•	Z	C
		ABSOLUTE	6D	4	3							
		ZERO PAGE	65	3	2							
		(IND,X)	61	6	2							
		(IND),Y	71	5	2							
		ZERO PAGE,X	75	4	2							
		ABS,X	7D	4	3							
		ABS,Y	79	4	3							
AND	A & M -> A (1)	IMMEDIATE	29	2	2	N	•	•	•	•	Z	•
		ABSOLUTE	2D	4	3							
		ZERO PAGE	25	3	2							
		(IND,X)	21	6	2							
		(IND),Y	31	5	2							
		ZERO PAGE,X	35	4	2							
		ABS,X	3D	4	3							
		ABS,Y	39	4	3							
ASL	C <- [7 0] <- 0	ABSOLUTE	0E	6	3	N	•	•	•	•	Z	C
		ZERO PAGE	06	5	2							
		ACCUM.	0A	2	1							
		ZERO PAGE,X	16	6	2							
		ABS,X	1E	7	3							
BCC	Branch on C=0 (2)	RELATIVE	90	2	2	•	•	•	•	•	•	•
BCS	Branch on C=1 (2)	RELATIVE	B0	2	2	•	•	•	•	•	•	•
BEQ	Branch on Z=1 (2)	RELATIVE	F0	2	2	•	•	•	•	•	•	•
BIT	A & M	ABSOLUTE	2C	4	3	M7	M6	•	•	•	Z	•
		ZERO PAGE	24	3	2							
BMI	Branch on N=1 (2)	RELATIVE	30	2	2	•	•	•	•	•	•	•
BNE	Branch on Z=0 (2)	RELATIVE	D0	2	2	•	•	•	•	•	•	•
BPL	Branch on N=0 (2)	RELATIVE	10	2	2	•	•	•	•	•	•	•
BRK	Break	IMPLIED	00	7	1	•	•	1	•	1	•	•
BVC	Branch on V=0 (2)	RELATIVE	50	2	2	•	•	•	•	•	•	•

Instructions Mnemonic	Operation	Addressing Mode	OP	N	#	N	V	B	D	I	Z	C
BVS	Branch on V=1 (2)	RELATIVE	70	2	2	•	•	•	•	•	•	•
CLC	0 -> C	IMPLIED	18	2	1	•	•	•	•	•	•	0
CLD	0 -> D	IMPLIED	D8	2	1	•	•	•	0	•	•	•
CLI	0 -> I	IMPLIED	58	2	1	•	•	•	•	0	•	•
CLV	0 -> V	IMPLIED	B8	2	1	•	0	•	•	•	•	•
CMP	A - M (1)	IMMEDIATE ABSOLUTE ZERO PAGE (IND,X) (IND),Y ZERO PAGE,X ABS,X ABS,Y	C9 CD C5 C1 D1 D5 DD D9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N	•	•	•	•	Z	C
CPX	X - M	IMMEDIATE ABSOLUTE ZERO PAGE	E0 EC E4	2 4 3	2 3 2	N	•	•	•	•	Z	C
CPY	Y - M	IMMEDIATE ABSOLUTE ZERO PAGE	C0 CC C4	2 4 3	2 3 2	N	•	•	•	•	Z	C
DEC	M - 1 -> M	ABSOLUTE ZERO PAGE ZERO PAGE,X ABS,X	CE C6 D6 DE	6 5 6 7	3 2 2 3	N	•	•	•	•	Z	•
DEX	X - 1 -> X	IMPLIED	CA	2	1	N	•	•	•	•	Z	•
DEY	Y - 1 -> Y	IMPLIED	88	2	2	N	•	•	•	•	Z	•

Instructions Mnemonic	Operation	Addressing Mode	OP	N	#	N	V	B	D	I	Z	C
EOR	A @ M -> A (1)	IMMEDIATE	49	2	2	N	.	.	.	.	Z	.
		ABSOLUTE	4D	4	3							
		ZERO PAGE	45	3	2							
		(IND,X)	41	6	2							
		(IND),Y	51	5	2							
		ZERO PAGE,X	55	4	2							
		ABS,X	5D	4	3							
		ABS,Y	59	4	3							
INC	M + 1 -> M	ABSOLUTE	EE	6	3	N	.	.	.	.	Z	.
		ZERO PAGE	E6	5	2							
		ZERO PAGE,X	F6	6	2							
		ABS,X	FE	7	3							
INX	X + 1 -> X	IMPLIED	E8	2	1	N	.	.	.	.	Z	.
INY	Y + 1 -> Y	IMPLIED	C8	2	1	N	.	.	.	.	Z	.
JMP	Jump to new loc.	ABSOLUTE	4C	3	3	.	.	.	.	.	.	.
		INDIRECT	6C	5	3							
JSR	Jump subr.	ABSOLUTE	20	6	3	.	.	.	.	.	.	.
LDA	M -> A (1)	IMMEDIATE	A9	2	2	N	.	.	.	.	Z	.
		ABSOLUTE	AD	4	3							
		ZERO PAGE	A5	3	2							
		(IND,X)	A1	6	2							
		(IND),Y	B1	5	2							
		ZERO PAGE,X	B5	4	2							
		ABS,X	BD	4	3							
		ABS,Y	B9	4	3							
LDX	M -> X (1)	IMMEDIATE	A2	2	2	N	.	.	.	.	Z	.
		ABSOLUTE	AE	4	3							
		ZERO PAGE	A6	3	2							
		ABS,Y	BE	4	3							
		ZERO PAGE,Y	B6	4	2							

Instructions Mnemonic	Operation	Addressing Mode	OP	N	#	N	V	B	D	I	Z	C
LDY	M -> Y (1)	IMMEDIATE	A0	2	2	N	•	•	•	•	Z	•
		ABSOLUTE	AC	4	3							
		ZERO PAGE	A4	3	2							
		ZERO PAGE,X	B4	4	2							
		ABS,X	BC	4	3							
LSR	0 -> [7 0] -> C	ABSOLUTE	4E	6	3	0	•	•	•	•	Z	C
		ZERO PAGE	46	5	2							
		ACCUM.	4A	2	1							
		ZERO PAGE,X	56	6	2							
		ABS,X	5E	7	3							
NOP	No Operation	IMPLIED	EA	2	1	•	•	•	•	•	•	•
ORA	A V M -> A (1)	IMMEDIATE	09	2	2	N	•	•	•	•	Z	•
		ABSOLUTE	0D	4	3							
		ZERO PAGE	05	3	2							
		(IND,X)	01	6	2							
		(IND),Y	11	5	2							
		ZERO PAGE,X	15	4	2							
		ABS,X	1D	4	3							
		ABS,Y	19	4	3							
PHA	A -> Ms SP-1 -> SP	IMPLIED	48	3	1	•	•	•	•	•	•	•
PHP	P -> Ms SP-1 -> SP	IMPLIED	08	3	1	•	•	•	•	•	•	•
PLA	SP+1 -> SP Ms -> A	IMPLIED	68	4	1	N	•	•	•	•	Z	•
PLP	SP+1 -> SP Ms -> P	IMPLIED	28	4	1	(Restored)						
ROL	<- [7 0] <- [C] <-	ABSOLUTE	2E	6	3	N	•	•	•	•	Z	C
		ZERO PAGE	26	5	2							
		ACCUM.	2A	2	1							
		ZERO PAGE,X	36	6	2							
		ABS,X	3E	7	3							
ROR	-> [C] -> [7 0] ->	ABSOLUTE	6E	6	3	N	•	•	•	•	Z	C
		ZERO PAGE	66	5	2							
		ACCUM.	6A	2	1							
		ZERO PAGE,X	76	6	2							
		ABS,X	7E	7	3							

Instructions Mnemonic	Operation	Addressing Mode	OP	N	#	N	V	B	D	I	Z	C
RTI	Return from intr.	IMPLIED	40	6	1	(Restored)						
RTS	Return from subr.	IMPLIED	60	6	1	•	•	•	•	•	•	•
SBC	A - M - $\bar{C}$ -> A (1)	IMMEDIATE	E9	2	2	N	V	•	•	•	Z	(3)
		ABSOLUTE	ED	4	3							
		ZERO PAGE	E5	3	2							
		(IND,X)	E1	6	2							
		(IND),Y	F1	5	2							
		ZERO PAGE,X	F5	4	2							
		ABS,X	FD	4	3							
		ABS,Y	F9	4	3							
SEC	1 -> C	IMPLIED	38	2	1	•	•	•	•	•	•	1
SED	1 -> D	IMPLIED	F8	2	1	•	•	•	1	•	•	•
SEI	1 -> I	IMPLIED	78	2	1	•	•	•	•	1	•	•
STA	A -> M	ABSOLUTE	8D	4	3	•	•	•	•	•	•	•
		ZERO PAGE	85	3	2							
		(IND,X)	81	6	2							
		(IND),Y	91	6	2							
		ZERO PAGE,X	95	4	2							
		ABS,X	9D	5	3							
		ABS,Y	99	5	3							
STX	X -> M	ABSOLUTE	8E	4	3	•	•	•	•	•	•	•
		ZERO PAGE	86	3	2							
		ZERO PAGE,Y	96	4	2							
STY	Y -> M	ABSOLUTE	8C	4	3	•	•	•	•	•	•	•
		ZERO PAGE	84	3	2							
		ZERO PAGE,X	94	4	2							
TAX	A -> X	IMPLIED	AA	2	1	N	•	•	•	•	Z	•
TAY	A -> Y	IMPLIED	A8	2	1	N	•	•	•	•	Z	•
TSX	SP -> X	IMPLIED	BA	2	1	N	•	•	•	•	Z	•
TXA	X -> A	IMPLIED	8A	2	1	N	•	•	•	•	Z	•

Instructions Mnemonic	Operation	Addressing Mode	OP	N	#	Condition Codes N	V	B	D	I	Z	C
TXS	X -> SP	IMPLIED	9A	2	1	•	•	•	•	•	•	•
TYA	Y -> A	IMPLIED	98	2	1	N	•	•	•	•	Z	•

(1): Add 1 to "N" if page boundary is crossed.
(2): Add 1 to "N" if branch occurs to same page.
Add 2 to "N" if branch occurs to different page.
(3): Carry Not = Borrow.

X: Index X   Y: Index Y   A: Accumulator   SP: Stack Pointer
M: Memory per effective address   Ms: Memory per stack pointer
+: Add   -: Subtract   &: AND   V: OR   ∂: Exclusive OR
M7: Memory bit 7   M6: Memory bit 6   N: No.Cycles   #: No.Bytes

APPENDIX B

SUMMARY OF CYCLE BY CYCLE OPERATION OF INSTRUCTIONS

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus

Immediate

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
ADC CPY LDY		T0	1	PC	1	OP CODE
AND EOR ORA	2	T1	0	PC + 1	1	Data
CMP LDA SBC		T0	1	PC + 2	1	OP CODE(Next)
CPX LDX						

Absolute

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
JMP		T0	1	PC	1	OP CODE
	3	T1	0	PC + 1	1	ADL
		T2	0	PC + 2	1	ADH
		T0	1	ADH,ADL	1	OP CODE(Next)
ADC CPY ORA		T0	1	PC	1	OP CODE
AND EOR SBC		T1	0	PC + 1	1	ADL
BIT LDA	4	T2	0	PC + 2	1	ADH
CMP LDX		T3	0	ADH,ADL	1	Data
CPX LDY		T0	1	PC + 3	1	OP CODE(Next)
STA		T0	1	PC	1	OP CODE
STX		T1	0	PC + 1	1	ADL
STY	4	T2	0	PC + 2	1	ADH
		T3	0	ADH,ADL	0	Data
		T0	1	PC + 3	1	OP CODE(Next)
ASL		T0	1	PC	1	OP CODE
DEC		T1	0	PC + 1	1	ADL
INC		T2	0	PC + 2	1	ADH
LSR	6	T3	0	ADH,ADL	1	Data
ROL		T4	0	ADH,ADL	0	Data
ROR		T5	0	ADH,ADL	0	Modified Data
		T0	1	PC + 3	1	OP CODE(Next)

Absolute -- Continued

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
JSR	6	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	ADL
		T2	0	01,SP	1	Data(Discarded)
		T3	0	01,SP	0	PCH(PC + 2)
		T4	0	01,SP - 1	0	PCL(PC + 2)
		T5	0	PC + 2	1	ADH
		T0	1	ADH,ADL	1	OP CODE(Next)

Zero Page

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
ADC CPX LDX AND CPY LDY BIT EOR ORA CMP LDA SBC	3	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	ADL
		T2	0	00,ADL	1	Data
		T0	1	PC + 2	1	OP CODE(Next)
STA STX STY	3	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	ADL
		T2	0	00,ADL	0	Data
		T0	1	PC + 2	1	OP CODE(Next)
ASL DEC INC LSR ROL ROR	5	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	ADL
		T2	0	00,ADL	1	Data
		T3	0	00,ADL	0	Data
		T4	0	00,ADL	0	Modified Data
		T0	1	PC + 2	1	OP CODE(Next)

Accumulator

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
ASL LSR ROL ROR	2	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	OP CODE(Discarded)
		T0	1	PC + 1	1	OP CODE(Next)

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
Implied						
CLC INX TAX CLD INY TAY CLI NOP TSX CLV SEC TXA DEX SED TXS DEY SEI TYA	2	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	OP CODE(Discarded)
		T0	1	PC + 1	1	OP CODE(Next)
PHA PHP	3	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	OP CODE(Discarded)
		T2	0	01,SP	0	Data
		T0	1	PC + 1	1	OP CODE(Next)
PLA PLP	4	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	OP CODE(Discarded)
		T2	0	01,SP	1	Data(Discarded)
		T3	0	01,SP + 1	1	Data
		T0	1	PC + 1	1	OP CODE(Next)
RTI	6	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	OP CODE(Discarded)
		T2	0	01,SP	1	Data(Discarded)
		T3	0	01,SP + 1	1	Data(P)
		T4	0	01,SP + 2	1	Data(PCL)
		T5	0	01,SP + 3	1	Data(PCH)
		T0	1	PCH,PCL	1	OP CODE(Next)
RTS	6	T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	Data(Discarded)
		T2	0	01,SP	1	Data(Discarded)
		T3	0	01,SP + 1	1	PCL
		T4	0	01,SP + 2	1	PCH
		T5	0	PCH,PCL	1	Data(Discarded)
		T0	1	PCH,PCL + 1	1	OP CODE(Next)

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
Implied -- Continued						
BRK		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	Data(Discarded)
		T2	0	01,SP	0	PCH(PC + 2)
	7	T3	0	01,SP − 1	0	PCL(PC + 2)
		T4	0	01,SP − 2	0	P
		T5	0	FFFE	1	ADL
		T6	0	FFFF	1	ADH
		T0	1	ADH,ADL	1	OP CODE(Next)
Indexed Indirect						
ADC		T0	1	PC	1	OP CODE
AND		T1	0	PC + 1	1	BAL
CMP		T2	0	00,BAL	1	Data(Discarded)
EOR	6	T3	0	00,BAL + X	1	ADL
LDA		T4	0	00,BAL+X+1	1	ADH
ORA		T5	0	ADH,ADL	1	Data
SBC		T0	1	PC + 2	1	OP CODE(Next)
STA		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	BAL
		T2	0	00,BAL	1	Data(Discarded)
	6	T3	0	00,BAL + X	1	ADL
		T4	0	00,BAL+X+1	1	ADH
		T5	0	ADH,ADL	0	Data
		T0	1	PC + 2	1	OP CODE(Next)
Indirect Indexed						
ADC SBC		T0	1	PC	1	OP CODE
AND		T1	0	PC + 1	1	IAL
CMP	5	T2	0	00,IAL	1	BAL
EOR		T3	0	00,IAL + 1	1	BAH
LDA		T4	0	BAH,BAL + Y	1	Data
ORA		T0	1	PC + 2	1	OP CODE(Next)

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus

Indirect Indexed -- Continued

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
ADC		T0	1	PC	1	OP CODE
AND		T1	0	PC + 1	1	IAL
CMP		T2	0	00,IAL	1	BAL
EOR	6*	T3	0	00,IAL + 1	1	BAH
LDA		T4	0	BAH,BAL + Y	1	Data(Discarded)
ORA		T5	0	BAH+1,BAL+Y	1	Data
SBC		T0	1	PC + 2	1	OP CODE(Next)
STA		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	IAL
		T2	0	00,IAL	1	BAL
	6	T3	0	00,IAL + 1	1	BAH
		T4	0	BAH,BAL + Y	1	Data(Discarded)
		T5	0	BAH+C,BAL+Y	0	Data
		T0	1	PC + 2	1	OP CODE(Next)

Zero Page Indexed, X

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
ADC LDY		T0	1	PC	1	OP CODE
AND ORA		T1	0	PC + 1	1	BAL
CMP SBC	4	T2	0	00,BAL	1	Data(Discarded)
EOR		T3	0	00,BAL + X	1	Data
LDA		T0	1	PC + 2	1	OP CODE(Next)
STA		T0	1	PC	1	OP CODE
STY		T1	0	PC + 1	1	BAL
	4	T2	0	00,BAL	1	Data(Discarded)
		T3	0	00,BAL + X	0	Data
		T0	1	PC + 2	1	OP CODE(Next)
ASL		T0	1	PC	1	OP CODE
DEC		T1	0	PC + 1	1	BAL
INC		T2	0	00,BAL	1	Data(Discarded)
LSR	6	T3	0	00,BAL + X	1	Data
ROL		T4	0	00,BAL + X	0	Data
ROR		T5	0	00,BAL + X	0	Modified Data
		T0	1	PC + 2	1	OP CODE(Next)

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus

Zero Page Indexed, Y

LDX		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	BAL
	4	T2	0	00,BAL	1	Data(Discarded)
		T3	0	00,BAL + Y	1	Data
		T0	1	PC + 2	1	OP CODE(Next)
STX		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	BAL
	4	T2	0	00,BAL	1	Data(Discarded)
		T3	0	00,BAL + Y	0	Data
		T0	1	PC + 2	1	OP CODE(Next)

Indirect

JMP		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	IAL
	5	T2	0	PC + 2	1	IAH
		T3	0	IAH,IAL	1	ADL
		T4	0	IAH,IAL + 1	1	ADH
		T0	1	ADH,ADL	1	OP CODE(Next)

Absolute Indexed, X

ADC LDY		T0	1	PC	1	OP CODE
AND ORA		T1	0	PC + 1	1	BAL
CMP SBC	4	T2	0	PC + 2	1	BAH
EOR		T3	0	BAH,BAL + X	1	Data
LDA		T0	1	PC + 3	1	OP CODE(Next)
ADC ORA		T0	1	PC	1	OP CODE
AND SBC		T1	0	PC + 1	1	BAL
CMP	5*	T2	0	PC + 2	1	BAH
EOR		T3	0	BAH,BAL + X	1	Data(Discarded)
LDA		T4	0	BAH+1,BAL+X	1	Data
LDY		T0	1	PC + 3	1	OP CODE(Next)

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus

Absolute Indexed, X -- <u>Continued</u>

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
STA		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	BAL
	5	T2	0	PC + 2	1	BAH
		T3	0	BAH,BAL + X	1	Data(Discarded)
		T4	0	BAH+C,BAL+X	0	Data
		T0	1	PC + 3	1	OP CODE(Next)
ASL		T0	1	PC	1	OP CODE
DEC		T1	0	PC + 1	1	BAL
INC		T2	0	PC + 2	1	BAH
LSR	7	T3	0	BAH,BAL + X	1	Data(Discarded)
ROL		T4	0	BAH+C,BAL+X	1	Data
ROR		T5	0	BAH+C,BAL+X	0	Data
		T6	0	BAH+C,BAL+X	0	Modified Data
		T0	1	PC + 3	1	OP CODE(Next)

Absolute Indexed, Y

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
ADC LDX		T0	1	PC	1	OP CODE
AND ORA		T1	0	PC + 1	1	BAL
CMP SBC	4	T2	0	PC + 2	1	BAH
EOR		T3	0	BAH,BAL + Y	1	Data
LDA		T0	1	PC + 3	1	OP CODE(Next)
ADC ORA		T0	1	PC	1	OP CODE
AND SBC		T1	0	PC + 1	1	BAL
CMP	5*	T2	0	PC + 2	1	BAH
EOR		T3	0	BAH,BAL + Y	1	Data(Discarded)
LDA		T4	0	BAH+1,BAL+Y	1	Data
LDX		T0	1	PC + 3	1	OP CODE(Next)
STA		T0	1	PC	1	OP CODE
		T1	0	PC + 1	1	BAL
	5	T2	0	PC + 2	1	BAH
		T3	0	BAH,BAL + Y	1	Data(Discarded)
		T4	0	BAH+C,BAL+Y	0	Data
		T0	1	PC + 3	1	OP CODE(Next)

Address Modes & Instructions	Cycles #	Cycle #	SYNC Line	Address Bus	R/W Line	Data Bus
Relative						
BCC BMI BVC BCS BNE BVS BEQ BPL	2!	T0 T1 T0	1 0 1	PC PC + 1 PC + 2	1 1 1	OP CODE Offset OP CODE(Next)
BCC BNE BCS BPL BEQ BVC BMI BVS	3@	T0 T1 T2 T0	1 0 0 1	PC PC + 1 PC + 2 PC+2+Offset	1 1 1 1	OP CODE Offset OP CODE(Discarded) OP CODE(Next)
BCC BPL BCS BVC BEQ BVS BMI BNE	4$	T0 T1 T2 T3 T0	1 0 0 0 1	PC PC + 1 PC + 2 PC+2+Offset PC+2+Offset	1 1 1 1 1	OP CODE Offset OP CODE(Discarded) Data(Discarded) OP CODE(Next)

* --- With page crossing.
! --- Branch not taken.
@ --- Branch taken without page crossing.
$ --- Branch taken with page crossing.

APPENDIX C

INTERNAL OPERATION SEQUENCES OF INSTRUCTIONS

#1 - ADC using Immediate Addressing Mode(2)

```
T0:          IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.

T0[OVLP]:    IADBUS = INR; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS);
             OP = \0,0,0,1\; ARGBUS = AC; IDBUS = DL;
             PS = P[4],P[7];
             ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
             COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
             P[0] <= ALUOUT[0]; P[1] * ((ARGBUS[0] & IDBUS[0]
             & ^ALUOUT[0]) + (^ARGBUS[0] & ^IDBUS[0] &
             ALUOUT[0])) <= \1\; P[1] * ^((ARGBUS[0]] &
             IDBUS[0] & ^ALUOUT[0]) + (^ARGBUS[0] & ^IDBUS[0]
             & ALUOUT[0])) <= \0\; P[6] <= ^(+/ALUOUT);
             P[7] <= COUT.

T1[OVLP]:    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR;
             IDBUS = ALUREG; AC <= IDBUS.
```

#2 - JMP using Absolute Addressing Mode(3)

```
T0:          IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.

T2:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); IDBUS = DL; TR <= IDBUS.

T0[OVLP]:    IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
             RW = \1\; INR <= INC(IADBUS).

T1[OVLP]:    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.
```

#3 - AND using Absolute Addressing Mode(4)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); IDBUS = DL; TR <= IDBUS.

T3:        IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
           RW = \1\; PC <= INR.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS);
           OP = \0,0,1,0\; ARGBUS = AC; IDBUS = DL;
           PS = P[4],P[7];
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
           ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
           P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR;
           IDBUS = ALUREG; AC <= IDBUS.
```

#4 - STA using Absolute Addressing Mode(4)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); IDBUS = DL; TR <= IDBUS.

T3:        IADBUS = DL,TR; SYNC = \0\; RW = \0\;
           IDBUS = AC; DBUS = IDBUS; PC <= INR.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#5 - ASL using Absolute Addressing Mode(6)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); IDBUS = DL; TR <= IDBUS.

T3:        IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
           RW = \1\; INR <= IADBUS; PC <= INR.

T4:        IADBUS = INR; SYNC = \0\; RW = \0\;
           OP = \0,1,1,1\; ARGBUS = \0,0,0,0,0,0,0,0\;
           IDBUS = DL; PS = P[4],P[7]; DBUS = IDBUS;
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
           COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
           P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT);
           P[7] <= COUT.

T5:        IADBUS = INR; SYNC = \0\; RW = \0\;
           IDBUS = ALUREG; DBUS = IDBUS.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).
```

ASL -- Continued

```
T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#6 - JSR using Absolute Addressing Mode(6)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = \0,0,0,0,0,0,0,1\,SP; DL <= DBUS;
           SYNC = \0\; RW = \1\; INR <= IADBUS; IDBUS = DL;
           TR <= IDBUS; PC <= INR.

T3:        IADBUS = INR; SYNC = \0\; RW = \0\;
           INR[8:15] <= DEC[8:15](IADBUS); IDBUS = PC[0:7];
           DBUS = IDBUS.

T4:        IADBUS = INR; SYNC = \0\; RW = \0\;
           INR[8:15] <= DEC[8:15](IADBUS);
           IDBUS = PC[8:15]; DBUS = IDBUS.

T5:        IADBUS = PC; DL <= DBUS; SYNC = \0\; RW = \1\;
           PC <=INR.

T0[OVLP]:  IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
           RW = \1\; INR <= INC(IADBUS);
           IDBUS = PC[8:15]; SP <= IDBUS.

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#7 - BIT using Zero Page Addressing Mode(3)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; PC <= INR.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,0,1,0\; ARGBUS = AC; IDBUS = DL;
            PS = P[4],P[7];
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= IDBUS[0]; P[1] <= IDBUS[1];
            P[6] <= ^(+/ALUOUT).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#8 - STX using Zero Page Addressing Mode(3)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; SYNC = \0\;
            RW = \0\; IDBUS = X; DBUS = IDBUS; PC <= INR.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#9 - DEC using Zero Page Addressing Mode(5)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR <= IADBUS; PC <= INR.

T3:         IADBUS = INR; SYNC = \0\; RW = \0\;
            OP = \0,0,0,0\; ARGBUS = \1,1,1,1,1,1,1,1\;
            IDBUS = DL; PS = P[4],P[7]; DBUS = IDBUS;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T4:         IADBUS = INR; SYNC = \0\; RW = \0\;
            IDBUS = ALUREG; DBUS = IDBUS.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#10 - LSR using Accumulator Addressing Mode(2)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS);
           OP = \1,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,0\;
           IDBUS = AC; PS = P[4],P[7];
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
           COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
           P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT);
           P[7] <= COUT.

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR;
           IDBUS = ALUREG; AC <= IDBUS.
```

#11 - CLC using Implied Addressing Mode(2)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS);
           P[7] <= \0\.

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#12 - PHA using Implied Addressing Mode(3)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
           RW = \0\; IDBUS = AC; DBUS = IDBUS;
           INR[0:7] <= IADBUS[0:7];
           INR[8:15] <= DEC[8:15](IADBUS).

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS);
           PC <= INR.

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR;
           IDBUS = PC[8:15]; SP <= IDBUS.
```

#13 - PLP using Implied Addressing Mode(4)

```
T0:          IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.

T2:          IADBUS = \0,0,0,0,0,0,0,1\,SP; DL <= DBUS;
             SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
             INR[8:15] <= INC[8:15](IADBUS).

T3:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\.

T0[OVLP]:    IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS);
             IDBUS = DL; P <= IDBUS; PC <= INR.

T1[OVLP]:    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR;
             IDBUS = PC[8:15]; SP <= IDBUS.
```

#14 - RTI using Implied Addressing Mode(6)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = \0,0,0,0,0,0,0,1\,SP; DL <= DBUS;
           SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
           INR[8:15] <= INC[8:15](IADBUS).

T3:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR[8:15] <= INC[8:15](IADBUS).

T4:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR[8:15] <= INC[8:15](IADBUS);
           IDBUS = DL; P <= IDBUS.

T5:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           IDBUS = DL; TR <= IDBUS; PC <= INR.

T0[OVLP]:  IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
           RW = \1\; INR <= INC(IADBUS);
           IDBUS = PC[8:15]; SP <= IDBUS.

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#15 - RTS using Implied Addressing Mode(6)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,1\,SP; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= INC[8:15](IADBUS).

T3:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR[8:15] <= INC[8:15](IADBUS).

T4:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS.

T5:         IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
            RW = \1\; INR <= INC(IADBUS); PC <= INR.

T0[OVLP]:   IADBUS = INR; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            IDBUS = PC[8:15]; SP <= IDBUS.

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#16 - BRK using Implied Addressing Mode(7)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
           RW = \0\; IDBUS = INR[0:7]; DBUS = IDBUS;
           INR[0:7] <= IADBUS[0:7];
           INR[8:15] <= DEC[8:15](IADBUS); PC <= INR.

T3:        IADBUS = INR; SYNC = \0\; RW = \0\;
           IDBUS = PC[8:15]; DBUS = IDBUS; P[3] <= \1\;
           INR[8:15] <= DEC[8:15](IADBUS).

T4:        IADBUS = INR; SYNC = \0\; RW = \0\; IDBUS = P;
           DBUS = IDBUS; INR[8:15] <= DEC[8:15](IADBUS).

T5:        IADBUS = \1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0\;
           DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T6:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           IDBUS = DL; TR <= IDBUS; P[5] <= \1\.

T0[OVLP]:  IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
```

BRK -- <u>Continued</u>

```
            RW = \1\; INR <= INC(IADBUS);
            IDBUS = PC[8:15]; SP <= IDBUS.

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#17 - CMP using Indexed Indirect Addressing Mode(6)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
           SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
           OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
           IDBUS = X; PS = P[4],P[7]; PC <= INR;
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:        IADBUS = INR[0:7],ALUREG; DL <= DBUS;
           SYNC = \0\; RW = \1\;
           INR[8:15] <= INC[8:15](IADBUS).

T4:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           IDBUS = DL; TR <= IDBUS.

T5:        IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
           RW = \1\.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS);
           OP = \0,1,1,0\; ARGBUS = AC; IDBUS = DL;
           PS = P[4],P[7];
```

CMP -- Continued

```
            COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT);
            P[7] <= COUT.

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#18 - STA using Indexed Indirect Addressing Mode(6)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
            IDBUS = X; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\;
            INR[8:15] <= INC[8:15](IADBUS).

T4:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS.

T5:         IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
            RW = \0\; IDBUS = AC; DBUS = IDBUS.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#19 - EOR using Indirect Indexed Addressing Mode(5)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= INC[8:15](IADBUS); PC <= INR.

T3:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            OP = \0,0,0,0\; ARGBUS = Y; IDBUS = DL;
            PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T4:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,1,0,0\; ARGBUS = AC; IDBUS = DL;
            PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).
```

EOR -- Continued

```
T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR;
           IDBUS = ALUREG; AC <= IDBUS.
```

#20 - LDA using Indirect Indexed Addressing Mode(6)
[with page crossing]

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= INC[8:15](IADBUS); PC <= INR.

T3:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            OP = \0,0,0,0\; ARGBUS = Y; IDBUS = DL;
            PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T4:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\; INR[8:15] <= IADBUS[8:15];
            OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,1\;
            IDBUS = DL; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T5:         IADBUS = ALUREG,INR[8:15]; DL <= DBUS;
            SYNC = \0\; RW = \1\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
```

LDA -- <u>Continued</u>

```
            OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,0\;
            IDBUS = DL; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; AC <= IDBUS.
```

#21 - STA using Indirect Indexed Addressing Mode(6)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= INC[8:15](IADBUS); PC <= INR.

T3:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            OP = \0,0,0,0\; ARGBUS = Y; IDBUS = DL;
            PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T4:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\; OP = \0,0,0,0\; ARGBUS =
            (\0,0,0,0,0,0,0,0\ ! \0,0,0,0,0,0,0,1\) *
            (^C, C); IDBUS = DL; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            INR[8:15] <= IADBUS[8:15].

T5:         IADBUS = ALUREG,INR[8:15]; SYNC = \0\; RW =\0\;
            IDBUS = AC; DBUS = IDBUS.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
```

STA -- <u>Continued</u>

```
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#22 - LDY using Zero Page Indexed, X Addressing Mode(4)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
            IDBUS = X; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,0\;
            IDBUS = DL; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; Y <= IDBUS.
```

#23 - STY using Zero Page Indexed, X Addressing Mode(4)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
           SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
           OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
           IDBUS = X; PS = P[4],P[7]; PC <= INR;
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:        IADBUS = INR[0:7],ALUREG; SYNC = \0\; RW =\0\;
           IDBUS = Y; DBUS = IDBUS.

T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#24 - INC using Zero Page Indexed, X Addressing Mode(6)

```
T0:      IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
         INR <= INC(IADBUS).

T1:      IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
         INR <= INC(IADBUS); PC <= INR.

T2:      IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
         SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
         OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
         IDBUS = X; PS = P[4],P[7]; PC <= INR;
         ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:      IADBUS = INR[0:7],ALUREG; DL <= DBUS;
         SYNC = \0\; RW = \1\; INR <= IADBUS.

T4:      IADBUS = INR; SYNC = \0\; RW = \0\;
         DBUS = IDBUS; OP = \0,0,0,0\;
         ARGBUS = \0,0,0,0,0,0,0,1\; IDBUS = DL;
         PS = P[4],P[7];
         ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
         ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
         P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T5:      IADBUS = INR; SYNC = \0\; RW = \0\;
         IDBUS = ALUREG; DBUS = IDBUS.
```

INC -- Continued

```
T0[OVLP]:  IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1[OVLP]:  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.
```

#25 - LDX using Zero Page Indexed, Y Addressing Mode(4)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7] <= IADBUS[0:7];
            OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
            IDBUS = Y; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,0\;
            IDBUS = DL; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; X <= IDBUS.
```

#26 - STX using Zero Page Indexed, Y Addressing Mode(4)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,0\,DL; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[0:7]] <= IADBUS[0:7];
            OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
            IDBUS = Y; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; SYNC = \0\; RW = \0\;
            IDBUS = X; DBUS = IDBUS.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#27 - JMP using Indirect Addressing Mode(5)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS.

T3:         IADBUS = DL,TR; DL <= DBUS; SYNC = \0\;
            RW = \1\; INR <= INC(IADBUS).

T4:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS.

T0[OVLP]:   IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
            RW = \1\; INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#28 - ORA using Absolute Indexed, X Addressing Mode(4)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); OP = \0,0,0,0\; ARGBUS = X;
            IDBUS = DL; PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\; PC <= INR.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,0,1,1\; ARGBUS = AC; IDBUS = DL;
            PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; AC <= IDBUS.
```

#29 - SBC using Absolute Indexed, X Addressing Mode(5)
[with page crossing]

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); OP = \0,0,0,0\; ARGBUS = X;
            IDBUS = DL; PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\; INR[8:15] <= IADBUS[8:15];
            OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,1\;
            IDBUS = DL; PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T4:         IADBUS = ALUREG,INR[8:15]; DL <= DBUS;
            SYNC = \0\; RW = \1\.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,1,0,1\; ARGBUS = AC; IDBUS = DL;
            PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
```

SBC -- <u>Continued</u>

```
            COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[1] * ((ARGBUS[0] &
            IDBUS[0] & ^ALUOUT[0]) + (^ARGBUS[0] &
            ^IDBUS[0] & ALUOUT[0])) <= \1\; P[1] *
            ^((ARGBUS[0] & IDBUS[0] & ^ALUOUT[0]) +
            (^ARGBUS[0] & ^IDBUS[0] & ALUOUT[0])) <= \0\;
            P[6] <= ^(+/ALUOUT); P[7] <= COUT.

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; AC <= IDBUS.
```

#30 - STA using Absolute Indexed, X Addressing Mode(5)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); OP = \0,0,0,0\;
            ARGBUS = X; IDBUS = DL; PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\; INR[8:15] <= IADBUS[8:15];
            OP = \0,0,0,0\; ARGBUS = (\0,0,0,0,0,0,0\ !
            \0,0,0,0,0,0,0,1\) * (^C, C); IDBUS = DL;
            PS = P[4],P[7]; PC <= INR;
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T4:         IADBUS = ALUREG,INR[8:15]; SYNC = \0\; RW = \0\;
            IDBUS = AC; DBUS = IDBUS.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#31 - ROL using Absloute Indexed, X Addressing Mode(7)

```
T0:        IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
           INR <= INC(IADBUS).

T1:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); PC <= INR.

T2:        IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
           INR <= INC(IADBUS); OP = \0,0,0,0\; ARGBUS = X;
           IDBUS = DL; PS = P[4],P[7];
           C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:        IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
           RW = \1\; INR[8:15] <= IADBUS[8:15];
           OP = \0,0,0,0\; ARGBUS = (\0,0,0,0,0,0,0,0\ !
           \0,0,0,0,0,0,0,1\) * (^C, C); IDBUS = DL;
           PS = P[4],P[7]; PC <= INR;
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T4:        IADBUS = ALUREG,INR[8:15]; DL <= DBUS;
           SYNC = \0\; RW = \1\; INR <= IADBUS.

T5:        IADBUS = INR; SYNC = \0\; RW = \0\;
           OP = \1,0,0,1\; ARGBUS = \0,0,0,0,0,0,0,0\;
           IDBUS = DL; PS = P[4],P[7]; DBUS = IDBUS;
           ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
           COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
```

ROL -- <u>Continued</u>

```
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT);
            P[7] <= COUT.

T6:         IADBUS = INR; SYNC = \0\; RW = \0\;
            IDBUS = ALUREG; DBUS = IDBUS.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#32 - ADC using Absolute Indexed, Y Addressing Mode(4)

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); OP = \0,0,0,0\; ARGBUS = Y;
            IDBUS = DL; PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
            RW = \1\; PC <= INR.

T0[OVLP]:   IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS);
            OP = \0,0,0,1\; ARGBUS = AC; IDBUS = DL;
            PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
            COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[1] * ((ARGBUS[0] &
            IDBUS[0] & ^ALUOUT[0]) + (^ARGBUS[0] &
            ^IDBUS[0] & ALUOUT[0])) <= \1\; P[1] *
            ^((ARGBUS[0] & IDBUS[0] & ^ALUOUT[0]) +
            (^ARGBUS[0] & ^IDBUS[0] & ALUOUT[0])) <= \0\;
```

ADC -- Continued

```
            P[6] <= ^(+/ALUOUT); P[7] <= COUT.

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; AC <= IDBUS.
```

#33 - AND using Absolute Indexed, Y Addressing Mode(5)
[with page crossing]

```
T0:          IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.

T2:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); OP = \0,0,0,0\; ARGBUS = Y;
             IDBUS = DL; PS = P[4],P[7];
             C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:          IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
             RW = \1\; INR[8:15] <= IADBUS[8:15];
             OP = \0,0,0,0\; ARGBUS = \0,0,0,0,0,0,0,1\;
             IDBUS = DL; PS = P[4],P[7]; PC <= INR;
             ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T4:          IADBUS = ALUREG,INR[8:15]; DL <= DBUS;
             SYNC = \0\; RW = \1\.

T0[OVLP]:    IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS);
             OP = \0,0,1,0\; ARGBUS = AC; IDBUS = DL;
             PS = P[4],P[7];
             ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS);
```

AND -- <u>Continued</u>

```
            ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
            P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            IDBUS = ALUREG; AC <= IDBUS.
```

#34 - STA using Absolute Indexed, Y Addressing Mode(5)

```
T0:          IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.

T2:          IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUUS); OP = \0,0,0,0\; ARGBUS = Y;
             IDBUS = DL; PS = P[4],P[7];
             C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T3:          IADBUS = DL,ALUREG; DL <= DBUS; SYNC = \0\;
             RW = \1\; INR[8:15] <= IADBUS[8:15];
             OP = \0,0,0,0\; ARGBUS = (\0,0,0,0,0,0,0,0\ !
             \0,0,0,0,0,0,0,1\) * (^C, C); IDBUS = DL;
             PS = P[4],P[7]; PC <= INR;
             ALUREG <= ALU[1:8](OP; ARGBUS IDBUS; PS).

T4:          IADBUS = ALUREG,INR[8:15]; SYNC = \0\; RW = \0\;
             IDBUS = AC; DBUS = IDBUS.

T0[OVLP]:    IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
             INR <= INC(IADBUS).

T1[OVLP]:    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
             INR <= INC(IADBUS); PC <= INR.
```

#35 - BCC using Relative Addressing Mode(2)
[branch not taken]

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T0[OVLP]:   IADBUS = INR; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#36 - BCS using Relative Addressing Mode(3)
[branch taken without page crossing]

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            N <= DL[0]; OP = \0,0,0,0\;
            ARGBUS = IADBUS[8:15]; IDBUS = DL;
            PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS).

T0[OVLP]:   IADBUS = INR[0:7],ALUREG; IR <= DBUS;
            SYNC = \1\; RW = \1\; INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

#37 - BEQ using Relative Addressing Mode(4)
[branch taken with page crossing]

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            N <= DL[0]; OP = \0,0,0,0\;
            ARGBUS = IADBUS[8:15]; IDBUS = DL;
            PS = P[4],P[7];
            C,ALUREG <= ALU(OP; ARGBUS; IDBUS; PS);
            COUT = ALU[0](OP; ARGBUS; IDBUS; PS).

T3:         IADBUS = INR[0:7],ALUREG; DL <= DBUS;
            SYNC = \0\; RW = \1\; INR[8:15] <= IADBUS[8:15];
            OP = \0,0,0,0\; ARGBUS = (\0,0,0,0,0,0,0,1\ !
            \1,1,1,1,1,1,1,1\) * (^N & C, N & ^C);
            IDBUS = INR[0:7]; PS = P[4],P[7];
            ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).

T0[OVLP]:   IADBUS = ALUREG,INR[8:15]; IR <= DBUS;
            SYNC = \1\; RW = \1\; INR <= INC(IADBUS).

T1[OVLP]:   IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.
```

APPENDIX D

INTERNAL OPERATION SEQUENCES OF INPUTS

#1 - RES Input

```
T0:         IADBUS = PC; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS).

T1:         IADBUS = INR; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
            RW = \1\; INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= DEC[8:15](IADBUS).

T3:         IADBUS = INR; SYNC = \0\; RW = \1\;
            INR[8:15] <= DEC[8:15](IADBUS).

T4:         IADBUS = INR; SYNC = \0\; RW = \1\;
            INR[8:15] <= DEC[8:15](IADBUS).

T5:         IADBUS = \1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0\;
            DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T6:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS; P[5] <= \1\.

T0[OVLP]:   IADBUS = DL,TR; DL <= DBUS; SYNC = \1\;
            RW = \1\; INR <= INC(IADBUS);
            IDBUS = PC[8:15]; SP <= IDBUS.
```

#2 - IRQ Input

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= IADBUS.

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
            RW = \0\; IDBUS = PC[0:7]; DBUS = IDBUS;
            INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= DEC[8:15](IADBUS).

T3:         IADBUS = INR; SYNC = \0\; RW = \0\;
            IDBUS = PC[8:15]; DBUS = IDBUS;
            INR[8:15] <= DEC[8:15](IADBUS); P[3] <= \0\.

T4:         IADBUS = INR; SYNC = \0\; RW = \0\;
            IDBUS = P; DBUS = IDBUS;
            INR[8:15] <= DEC[8:15](IADBUS).

T5:         IADBUS = \1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0\;
            DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T6:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS; P[5] <= \1\.

T0[OVLP]:   IDBUS = DL,TR; IR <= DBUS; SYNC <= \1\;
```

IRQ Input -- Continued

```
            RW = \1\; INR <= INC(IADBUS);
            IDBUS = PC[8:15]; SP <= IDBUS.
```

#3 - RDY Input

```
BODY SEQUENCE: CLOCK.
        -
        -
        -
        -
ENDSEQUENCE
CONTROLREST(1);
        -
     CLOCK = PHI2 & ^ONE;
     ONE = ^RDY * (RW & PHI1);
        -
END.
```

#4 - NMI Input

```
T0:         IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
            INR <= IADBUS.

T1:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T2:         IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
            RW = \0\; IDBUS = PC[0:7]; DBUS = IDBUS;
            INR[0:7] <= IADBUS[0:7];
            INR[8:15] <= DEC[8:15](IADBUS).

T3:         IADBUS = INR; SYNC = \0\; RW  = \0\;
            IDBUS = PC[8:15]; DBUS = IDBUS;
            INR[8:15] <= DEC[8:15](IADBUS).

T4:         IADBUS = INR; SYNC = \0\; RW = \0\;
            IDBUS = P; DBUS = IDBUS;
            INR[8:15] <= DEC[8:15](IADBUS).

T5:         IADBUS = \1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0\;
            DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR.

T6:         IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            IDBUS = DL; TR <= IDBUS; P[5] <= \1\.

T0[OVLP]:   IADBUS = DL,TR; IR <= DBUS; SYNC = \1\;
```

NMI Input -- Continued

```
RW = \1\; INR <= INC(IADBUS);

IDBUS = PC[8:15]; SP <= IDBUS.
```

#5 - S.O. Input

```
ENDSEQUENCE
CONTROLRESET;

     P[1] * (^SO & PHI1) <= \1\;

END.
```

APPENDIX E

KARNAUGH MAP OF OPERATION CODES

IR[4]IR[5] = 00

IR[6]IR[7] = 00

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 BRK IMP	4 RTI IMP	C CPY IMM	8
01	1 BPL REL	5 BVC REL	D BNE REL	9 BCC REL
11	3 BMI REL	7 BVS REL	F BEQ REL	B BCS REL
10	2 JSR ABS	6 RTS IMP	E CPX IMM	A LDY IMM

0

IR[4]IR[5] = 01

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4	C CPY ZERO	8 STY ZERO
01	1	5	D	9 STY Z,X
11	3	7	F	B LDY Z,X
10	2 BIT ZERO	6	E CPX ZERO	A LDY ZERO

4

IR[4]IR[5] = 11

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4 JMP ABS	C CPY ABS	8 STY ABS
01	1	5	D	9
11	3	7	F	B LDY ABS,X
10	2 BIT ABS	6 JMP IND	E CPX ABS	A LDY ABS

C

IR[4]IR[5] = 10

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 PHP IMP	4 PHA IMP	C INY IMP	8 DEY IMP
01	1 CLC IMP	5 CLI IMP	D CLD IMP	9 TYA IMP
11	3 SEC IMP	7 SEI IMP	F SED IMP	B CLV IMP
10	2 PLP IMP	6 PLA IMP	E INX IMP	A TAY IMP

8

IR[6]IR[7] = 01

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ORA IND,X	4 EOR IND,X	C CMP IND,X	8 STA IND,X
01	1 ORA IND,Y	5 EOR IND,Y	D CMP IND,Y	9 STA IND,Y
11	3 AND IND,Y	7 ADC IND,Y	F SBC IND,Y	B LDA IND,Y
10	2 AND IND,X	6 ADC IND,X	E SBC IND,X	A LDA IND,X

1

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ORA ZERO	4 EOR ZERO	C CMP ZERO	8 STA ZERO
01	1 ORA Z,X	5 EOR Z,X	D CMP Z,X	9 STA Z,X
11	3 AND Z,X	7 ADC Z,X	F SBC Z,X	B LDA Z,X
10	2 AND ZERO	6 ADC ZERC	E SBC ZERO	A LDA ZERO

5

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ORA ABS	4 EOR ABS	C CMP ABS	8 STA ABS
01	1 ORA ABS,X	5 EOR ABS,X	D CMP ABS,X	9 STA ABS,X
11	3 AND ABS,X	7 ADC ABS,X	F SBC ABS,X	B LDA ABS,X
10	2 AND ABS	6 ADC ABS	E SBC ABS	A LDA ABS

D

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ORA IMM	4 EOR IMM	C CMP IMM	8
01	1 ORA ABS,Y	5 EOR ABS,Y	D CMP ABS,Y	9 STA ABS,Y
11	3 AND ABS,Y	7 ADC ABS,Y	F SBC ABS,Y	B LDA ABS,Y
10	2 AND IMM	6 ADC IMM	E SBC IMM	A LDA IMM

9

IR[6]IR[7] = 11

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4	C	8
01	1	5 Not	D Used	9
11	3	7	F	B
10	2	6	E	A

3

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4	C	8
01	1	5 Not	D Used	9
11	3	7	F	B
10	2	6	E	A

7

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4	C	8
01	1	5 Not	D Used	9
11	3	7	F	B
10	2	6	E	A

F

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4	C	8
01	1	5 Not	D Used	9
11	3	7	F	B
10	2	6	E	A

B

IR[6]IR[7] = 10

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0	4	C	8
01	1	5	D	9
11	3	7	F	B
10	2	6	E	A LDX IMM

2

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ASL ZERO	4 LSR ZERO	C DEC ZERO	8 STX ZERO
01	1 ASL Z,X	5 LSR Z,X	D DEC Z,X	9 STX Z,Y
11	3 ROL Z,X	7 ROR Z,X	F INC Z,X	B LDX Z,Y
10	2 ROL ZERO	6 ROR ZERO	E INC ZERO	A LDX ZERO

6

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ASL ABS	4 LSR ABS	C DEC ABS	6 STX ABS
01	1 ASL ABS,X	5 LSR ABS,X	D DEC ABS,X	9
11	3 ROL ABS,X	7 ROR ABS,X	F INC ABS,X	B LDX ABS,Y
10	2 ROL ABS	6 ROR ABS	E INC ABS	A LDX ABS

E

IR[2]IR[3] \ IR[0]IR[1]	00	01	11	10
00	0 ASL ACC	4 LSR ACC	C DEX IMP	8 TXA IMP
01	1	5	D	9 TXS IMP
11	3	7	F	B TSX IMP
10	2 ROL ACC	6 ROL ACC	E NOP IMP	A TAX IMP

A

APPENDIX F

6502 AHPL DESCRIPTION

```
AHPLMODULE:  MPU6502.
    MEMORY:  AC[8]; DL[8]; IR[8]; PC[16]; INR[16]; ALUREG[8];
             SP[8]; X[8]; Y[8]; TR[8]; P[8]; RESF; IRQF; NMIF; FF1;
             FF2; C; N; NMIFF; EXT.
    EXINPUTS:  RES; IRQ; NMI; RDY; SO; PHIO.
    OUTPUTS:  SYNC; RW; PHI1; PHI2; ADBUS[16];
              COUT; OP[4]; PS[2]; ALUOUT[8]; ONE; CLOCK; OSC; NM.
    BUSES:  IDBUS[8]; IADBUS[16]; ARGBUS[8].
    EXBUSES:  DBUS[8].
    CLUNITS:  ALU[9]   <: ALUNIT.
    CLUNITS:  INC[16]  <: INCR.
    CLUNITS:  DEC[16]  <: DECR.
BODY SEQUENCE:  CLOCK.
1           IADBUS = PC; SYNC = \1\; RW = \1\;
            INR <= INC(IADBUS); NMIF <= \0\; IRQF <= \0\.
2           IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
            INR <= INC(IADBUS); PC <= INR;
            EXT * (RESF + NMIF + IRQF) <= \1\;
            => (RESF + NMIF + IRQF) / (25).
3           NODELAY
            => (IR[6] & IR[7]) / (21).
4           NODELAY
            => (^IR[0] & ^IR[3] & ^IR[5] & ^IR[6] & ^IR[7]) / (25).
5           NODELAY
            => (IR[4] & ^IR[5] & ^IR[7]) / (10).
6           NODELAY
            => ((^IR[5] & ^IR[7]) + (^IR[3] & IR[4] & ^IR[5])) / (12).
7           NODELAY
            => (IR[4]) / (16).
8           NODELAY
            => (IR[5]) / (19).
9           NODELAY
            => (42).
10          NODELAY
            => ((((^IR[0] + IR[1]) & IR[3]) + (IR[0] & IR[1] & IR[2])) &
            IR[6]) / (21).
11          NODELAY
            => (30).
12          NODELAY
            => (IR[0] & ^IR[1] & ^IR[2] & ^IR[3]) / (21).
13          NODELAY
            => (IR[0] & ^IR[1] & IR[2] & ^IR[3]) / (31).
14          NODELAY
            => (IR[6]) / (21).
15          NODELAY
            => (31).
16          NODELAY
            => (IR[0] & ^IR[1] & ^IR[2] & IR[3] & ^IR[7]) / (21).
```

```
17      NODELAY
        => (((^IR[0] & ^IR[1] & ^IR[2]) + ((^IR[0] + IR[1]) &
        IR[3])) & ^IR[6] & ^IR[7]) / (21).
18      NODELAY
        => (35).
19      NODELAY
        => ((((^IR[0] + IR[1]) & IR[3]) + (^IR[0] & ^IR[2]) +
        (^IR[0] & IR[1])) & ^IR[6] & ^IR[7]) / (21).
20      NODELAY
        => (40).
21      IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
        INR <= (INC(IADBUS) ! IADBUS) * (^(NMIFF + NM + (^IRQ & ^P[5])),
        (NMIFF + NM + (^IRQ & ^P[5]))); IRQF * (^(NMIFF + NM) & ^IRQ &
        ^P[5]) <= \1\; NMIF * (NMIFF + NM) <= \1\; NMIFF <= \0\;
        => (2).
22      IADBUS = INR; IR <= DBUS; SYNC = \1\; RW = \1\;
        INR <= (INC(IADBUS) ! IADBUS) * (^(NMIFF + NM + (^IRQ & ^P[5])),
        (NMIFF + NM + (^IRQ & ^P[5]))); IRQF * (^(NMIFF + NM) & ^IRQ &
        ^P[5]) <= \1\; NMIF * (NMIFF + NM) <= \1\; NMIFF <= \0\;
        => (2).
23      IADBUS = DL,TR; IR <= DBUS; SYNC = \1\; RW = \1\;
        INR <= (INC(IADBUS) ! IADBUS) * (^(NMIFF + NM + (^IRQ & ^P[5])),
        (NMIFF + NM + (^IRQ & ^P[5]))); IRQF * (^(NMIFF + NM) & ^IRQ &
        ^P[5]) <= \1\; NMIF * (NMIFF + NM) <= \1\; NMIFF <= \0\;
        => (2).
24      IADBUS = (INR[0:7],ALUREG ! ALUREG,INR[8:15]) * (^(N @ C),
        N @ C); IR <= DBUS; SYNC = \1\; RW = \1\;
        INR <= (INC(IADBUS) ! IADBUS) * (^(NMIFF + NM + (^IRQ & ^P[5])),
        (NMIFF + NM + (^IRQ & ^P[5]))); IRQF * (^(NMIFF + NM) & ^IRQ &
        ^P[5]) <= \1\; NMIF * (NMIFF + NM) <= \1\; NMIFF <= \0\;
        => (2).
25      IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
        INR[0:7] <= IADBUS[0:7];
        INR[8:15] <= (IADBUS[8:15] ! DEC[8:15](IADBUS) !
        INC[8:15](IADBUS)) * (^IR[1] & IR[2] & ^IR[4] & ^EXT,
        EXT + ((^IR[1] + IR[4]) & ^IR[2]),
        ((IR[1] & ^IR[4]) + (IR[2] & IR[4])) & ^EXT);
        RW = ((IR[2] & ^EXT) + RESF + (IR[1] & ^IR[4] & ^EXT));
        DL * RW <= DBUS; DBUS = IDBUS * ^RW;
        IDBUS = (PC[0:7] ! INR[0:7] ! AC ! DL ! P) * (IRQF + NMIF,
        ^IR[1] & ^IR[2] & ^IR[4] & ^EXT, IR[1] & ^IR[2] & IR[4] &
        ^EXT, ^IR[1] & IR[2] & ^IR[4] & ^EXT, ^IR[1] & ^IR[2] &
        IR[4] & ^EXT); PC * (^IR[1] & ^IR[4] & ^EXT) <= INR;
        TR * (^IR[1] & IR[2] & ^IR[4] & ^EXT) <= IDBUS;
        => (^IR[2] & IR[4] & ^EXT, ^IR[2] & IR[4] & ^EXT) / (21, 47).
26      IADBUS = INR; SYNC = \0\; INR[8:15] <= (IADBUS[8:15] !
        INC[8:15](IADBUS) ! DEC[8:15](IADBUS)) * (IR[2] & IR[4] &
        ^EXT, IR[1] & ^IR[4] & ^EXT, (^IR[1] & ^IR[4]) + EXT);
```

```
      RW = ((((IR[1] & ^IR[4]) + (IR[2] & IR[4])) & ^EXT) + RESF);
      DL * RW <= DBUS; DBUS = IDBUS * ^RW;
      IDBUS = (PC[0:7] ! PC[8:15]) * (^IR[1] & IR[2] & ^IR[4] &
      ^EXT, IRQF + NMIF + (^IR[1] & ^IR[2] & ^EXT));
      P[3] * IRQF <= \0\; P[3] * (^IR[1] & ^IR[2] & ^EXT) <= \1\;
      => (IR[2] & IR[4] & ^EXT, IR[2] & IR[4] & ^EXT) / (21, 47).
27    IADBUS = INR; SYNC = \0\;
      INR[8:15] <= (INC[8:15](IADBUS) ! DEC[8:15](IADBUS)) *
      (IR[1] & ^IR[2] & ^EXT, ^IR[1] + EXT);
      RW = ((IR[1] & ^EXT) + RESF);
      DL * RW <= DBUS; DBUS = IDBUS * ^RW;
      IDBUS = (PC[8:15] ! DL ! P) * (^IR[1] & IR[2] & ^EXT,
      IR[1] & ^EXT, (^IR[1] & ^IR[2] & ^EXT) + IRQF + NMIF);
      P * (IR[1] & ^IR[2] & ^EXT) <= IDBUS;
      TR * (IR[1] & IR[2] & ^EXT) <= IDBUS.
28    IADBUS = (\1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0\ !
      \1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0\ !
      \1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0\ !
      PC ! INR ! DL,TR) * (NMIF, RESF, IRQF +
      (^IR[1] & ^IR[2] & ^EXT), ^IR[1] & IR[2] & ^EXT,
      IR[1] & ^IR[2] & ^EXT, IR[1] & IR[2] & ^EXT); SYNC = \0\;
      RW = \1\; DL <= DBUS; PC <= INR; IDBUS = DL;
      INR * ((^IR[1] & ^IR[2]) + (IR[1] & IR[2]) + EXT) <=
      INC(IADBUS); TR * (IR[1] & ^IR[2] & ^EXT) <= IDBUS;
      => (IR[1] & IR[2] & ^EXT, (IR[1] @ IR[2]) & ^EXT, (IR[1] +
      IR[2]) & ^EXT) / (22, 23, 48).
29    IADBUS = INR; SYNC = \0\; RW = \1\; DL <= DBUS; IDBUS = DL;
      TR <= IDBUS; P[5] <= \1\;
      RESF <= \0\; IRQF <= \0\; NMIF <= \0\; EXT <= \0\;
      => (23, 48).
30    NODELAY
      => (^IR[0] & ^IR[3], ^(^IR[0] & ^IR[3]), \1\) / (54, 49, 21).
31    NODELAY
      => (^IR[3], ^IR[3]) / (22, 56).
32    NODELAY
      => ((^IR[0] & ^IR[1] & ^IR[2] & P[0]) + (^IR[0] & ^IR[1] &
      IR[2] & ^P[0]) + (^IR[0] & IR[1] & ^IR[2] & P[1]) +
      (^IR[0] & IR[1] & IR[2] & ^P[1]) + (IR[0] & ^IR[1] &
      ^IR[2] & P[7]) + (IR[0] & ^IR[1] & IR[2] & ^P[7]) +
      (IR[0] & IR[1] & ^IR[2] & P[6]) + (IR[0] & IR[1] &
      IR[2] & ^P[6])) / (22).
33    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
      IDBUS = DL; N <= DL[0]; ARGBUS = IADBUS[8:15];
      OP = \0,0,0,0\; C <= ALU[0](OP; ARGBUS; IDBUS; PS);
      COUT = ALU[0](OP; ARGBUS; IDBUS; PS);
      => (^(DL[0] @ COUT)) / (24).
34    IADBUS = INR[0:7],ALUREG; DL <= DBUS;
      INR[8:15] <= IADBUS[8:15]; SYNC = \0\; RW = \1\;
```

```
    IDBUS = INR[0:7]; OP = \0,0,0,0\;
    ARGBUS = (\0,0,0,0,0,0,0,1\ ! \1,1,1,1,1,1,1,1\) *
    (^N & C, N & ^C);
    => (24).
35  IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
    INR <= INC(IADBUS); IDBUS = DL; OP = \0,0,0,0\; TR <= IDBUS;
    ARGBUS = (X ! Y) * ((IR[3] & IR[5] & ^IR[6]) + ((^IR[0] +
    IR[1]) & IR[3] & IR[6]), ^IR[5] + (IR[0] & ^IR[1] & IR[3] &
    IR[6])); C <= ALU[0](OP; ARGBUS; IDBUS; PS);
    => (^IR[0] & IR[1] & ^IR[2] & ^IR[3] & ^IR[6] & ^IR[7]) /
    (23).
36  IADBUS = (DL,ALUREG ! DL,TR) * (IR[3], ^IR[3]);
    SYNC = \0\; PC <= INR;
    RW = ^(IR[0] & ^IR[1] & ^IR[2] & ^IR[3]); DL * RW <= DBUS;
    DBUS = IDBUS * ^RW; IDBUS = (AC ! X ! Y ! DL) *
    (^RW & ^IR[6] & IR[7], ^RW & IR[6] & ^IR[7], ^RW & ^IR[6] &
    ^IR[7], IR[3]);
    INR <= (IADBUS ! INC(IADBUS)) * (IR[3] + IR[6], ^IR[0] &
    IR[1] & ^IR[6] & ^IR[7]); OP = \0,0,0,0\;
    ARGBUS = (\0,0,0,0,0,0,0,0\ ! \0,0,0,0,0,0,0,1\) *
    (IR[3] & ^C, IR[3] & C);
    => (IR[0] & ^IR[1] & ^IR[2] & ^IR[3], (IR[0] & ^IR[1] &
    IR[2] & ^IR[3]) + (^IR[0] & ^IR[1] & ^IR[6] & ^IR[7]) +
    (IR[0] & IR[1] & ^IR[6] & ^IR[7]) + ((^IR[0] + IR[1]) &
    ^IR[3] & IR[7]) + ((^(IR[0] & ^IR[1] & ^IR[2]) & IR[3] &
    ^IR[6]) + (IR[0] & ^IR[1] & IR[3] & IR[6])) & ^C, (^IR[0] +
    IR[1]) & ^IR[3] & IR[6], (^IR[0] + IR[1]) & IR[6]) /
    (21, 46, 54, 37).
37  IADBUS = (INR ! ALUREG,INR[8:15]) * (^IR[3], IR[3]);
    SYNC = \0\;
    RW = ^((^IR[3] & IR[6]) + (IR[0] & ^IR[1] & ^IR[2]));
    DL * RW <= DBUS; DBUS = IDBUS * ^RW; INR <= IADBUS;
    IDBUS = (AC ! DL) * (IR[0] & ^IR[1] & ^IR[2],
    ^IR[3] & ^IR[6]); TR <= IDBUS;
    => (IR[0] & ^IR[1] & ^IR[2], (^(IR[0] & ^IR[1] & ^IR[2]) &
    IR[3] & ^IR[6]) + (IR[0] & ^IR[1] & IR[6]), ^IR[3] & ^IR[6],
    (^IR[0] + IR[1]) & IR[3] & IR[6], (^IR[0] + IR[1]) & IR[6]) /
    (21, 46, 23, 54, 38).
38  IADBUS = INR; SYNC = \0\; RW = \0\; DBUS = IDBUS;
    => (^IR[3]) / (21).
39  IADBUS = INR; SYNC = \0\; RW = \0\; DBUS = IDBUS;
    => (21).
40  IADBUS = \0,0,0,0,0,0,0,0\,DL; SYNC = \0\; PC <= INR;
    RW = ^(IR[0] & ^IR[1] & ^IR[2] & ^IR[3]);
    DL * RW <= DBUS; DBUS = IDBUS * ^RW;
    IDBUS = (AC ! X ! Y) * (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] &
    IR[7], (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] & IR[6]) +
    (IR[3] & ^IR[6]) + ((^IR[0] + IR[1]) & IR[3] & IR[6]),
```

```
      (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] & ^IR[6] & ^IR[7]) +
      (IR[0] & ^IR[1] & IR[3] & IR[6]));
      INR <= IADBUS; OP = \0,0,0,0\; ARGBUS = IADBUS[8:15];
      => (IR[0] & ^IR[1] & ^IR[2] & ^IR[3], ((^IR[0] + IR[1]) &
      ^IR[3] & ^IR[6]) + (IR[0] & ^IR[1] & IR[2] & ^IR[3]),
      (^IR[0] + IR[1]) & ^IR[3] & IR[6], (^IR[0] + IR[1]) & IR[6])
      / (21, 46, 54, 41).
41    IADBUS = (INR ! INR[0:7],ALUREG) * (^IR[3], IR[3]);
      SYNC = \0\;
      RW = (((^IR[0] + IR[1]) & IR[3]) + (IR[2] & IR[3]));
      DL * RW <= DBUS; DBUS = IDBUS * ^RW;
      IDBUS = (AC ! X ! Y) * (IR[0] & ^IR[1] & ^IR[2] & IR[7],
      IR[0] & ^IR[1] & ^IR[2] & IR[6], ^IR[2] & ^IR[6] & ^IR[7]);
      INR <= IADBUS;
      => (IR[0] & ^IR[1] & ^IR[2] & IR[3], (IR[0] & ^IR[1] &
      IR[2] & IR[3]) + ((^IR[0] + IR[1]) & IR[3] & ^IR[6]),
      (^IR[0] + IR[1]) & IR[3] & IR[6], (^IR[0] + IR[1]) & IR[6]) /
      (21, 46, 54, 38).
42    IADBUS = \0,0,0,0,0,0,0,0\,DL; SYNC = \0\; RW = \1\;
      DL <= DBUS; PC <= INR; INR[0:7] <= IADBUS[0:7];
      INR[8:15] <= INC[8:15](IADBUS); IDBUS = X; OP = \0,0,0,0\;
      ARGBUS = IADBUS[8:15].
43    IADBUS = (INR ! INR[0:7],ALUREG) * (IR[3], ^IR[3]);
      SYNC = \0\; RW = \1\; DL <= DBUS;
      INR[8:15] <= INC[8:15](IADBUS); IDBUS = DL; ARGBUS = Y;
      OP = \0,0,0,0\; C <= ALU[0](OP; ARGBUS; IDBUS; PS).
44    IADBUS = (INR ! DL,ALUREG) * (^IR[3], IR[3]);
      SYNC = \0\; RW = \1\; DL <= DBUS; OP = \0,0,0,0\; IDBUS = DL;
      ARGBUS = (\0,0,0,0,0,0,0,0\ ! \0,0,0,0,0,0,0,1\) *
      (^C & IR[3], C & IR[3]);
      INR[8:15] <= IADBUS[8:15]; TR <= IDBUS;
      => (^(IR[0] & ^IR[1] & ^IR[2]) & IR[3] & ^C) / (46).
45    IADBUS = (ALUREG,INR[8:15] ! DL,TR) * (IR[3], ^IR[3]);
      SYNC = \0\;
      RW = ^(IR[0] & ^IR[1] & ^IR[2]);
      DL * RW <= DBUS; DBUS = IDBUS * ^RW;
      IDBUS = AC * (IR[0] & ^IR[1] & ^IR[2]);
      => (IR[0] & ^IR[1] & ^IR[2]) / (21).
46    NODELAY
      => (21, 56).
47    PC <= INR; IDBUS = DL; AC * (IR[1] & IR[2]) <= IDBUS;
      P * (^IR[1] & IR[2]) <= IDBUS; OP = \0,0,0,0\;
      ARGBUS = \0,0,0,0,0,0,0,0\;
      ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
      P[0] * (IR[1] & IR[2]) <= ALUOUT[0];
      P[6] * (IR[1] & IR[2]) <= ^(+/ALUOUT).
48    IDBUS = PC[8:15]; SP <= IDBUS;
      => (57).
```

```
49      P[7] * (^IR[0] & ^IR[1]) <= IR[2];
        P[5] * (^IR[0] & IR[1]) <= IR[2];
        P[4] * (IR[0] & IR[1] & IR[3]) <= IR[2];
        P[1] * (IR[0] & ^IR[1] & IR[2] & IR[3] & ^IR[6]) <= \0\;
        IDBUS = (AC ! X ! Y ! SP) * (^IR[1] & IR[2] & ^IR[3],
        (^IR[2] & IR[6]) + (IR[1] & IR[2] & ^IR[3]),
        (^IR[2] & ^IR[3] & ^IR[6]) + (IR[0] & ^IR[1] & ^IR[2] &
        ^IR[6]),  IR[2] & IR[3] & IR[6]); OP = \0,0,0,0\;
        ARGBUS = (\1,1,1,1,1,1,1,1\ ! \0,0,0,0,0,0,0,1\ !
        \0,0,0,0,0,0,0,0\) * ((^IR[1] & ^IR[2] &
        ^IR[3] & ^IR[6]) + (IR[1] & IR[6]), IR[1] &
        ^IR[3] & ^IR[6], (^IR[1] & IR[6]) + (IR[0] & ^IR[1] &
        ^IR[2] & IR[3]) + (^IR[1] & IR[2] & ^IR[3]));
        ALUOUT = ALU[1:8](OP; ARGBUS; IDBUS; PS);
        P[0] <= (^IR[3] + (IR[2] & IR[6]) + (IR[0] & ^IR[1] &
        ^IR[2] & ^IR[6])) <= ALUOUT[0];
        P[6] * (^IR[3] + (IR[2] & IR[6]) + (IR[0] & ^IR[1] &
        ^IR[2] & ^IR[6])) <= ^(+/ALUOUT);
        => ((IR[1] & IR[2] & ^IR[3]) + (IR[1] & IR[6]) + (IR[2] &
        IR[6]), (^IR[2] & ^IR[3] & ^IR[6]) + (^IR[1] & ^IR[3] &
        ^IR[6]), ^IR[2] & IR[3] & IR[6], ((^IR[0] + IR[1]) & IR[3]) +
        (IR[2] & IR[3] & ^IR[6])) / (51, 52, 53, 57).
50      IDBUS = ALUREG; AC <= IDBUS;
        => (57).
51      IDBUS = ALUREG; X <= IDBUS;
        => (57).
52      IDBUS = ALUREG; Y <= IDBUS;
        => (57).
53      IDBUS = ALUREG; SP <= IDBUS;
        => (57).
54      IDBUS = (AC ! DL) * (^IR[5], IR[5]);
        ARGBUS = (\0,0,0,0,0,0,0,1\ ! \1,1,1,1,1,1,1,1\) *
        (IR[0] & IR[2], IR[0] & ^IR[2]);
        OP = (\0,0,0,0\ ! \0,1,1,1\ ! \1,0,0,0\ ! \1,0,0,1\ !
        \1,0,1,0\) * (IR[0], ^IR[1] & ^IR[2], ^IR[0] & IR[1] &
        ^IR[2], ^IR[1] & IR[2], ^IR[0] & IR[1] & IR[2]);
        COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
        P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT);
        P[7] * ^IR[0] <= COUT;
        => (^IR[5]) / (50).
55      IDBUS = ALUREG;
        => (57).
56      IDBUS = DL; ARGBUS = (AC ! X ! Y ! \0,0,0,0,0,0,0,0\) *
        (^IR[0] + (IR[1] & IR[7]), IR[1] & IR[2] & ^IR[7],
        ^IR[2] & ^IR[7], IR[0] & ^IR[1]);
        OP = (\0,0,0,0\ ! \0,0,0,1\ ! \0,0,1,0\ ! \0,0,1,1\ !
        \0,1,0,0\ ! \0,1,0,1\ ! \0,1,1,0\) * (IR[0] & ^IR[1],
        ^IR[0] & IR[1] & IR[2], ^IR[0] & ^IR[1] & IR[2],
```

```
          ^IR[1] & ^IR[2], ^IR[0] & IR[1] & ^IR[2], IR[0] & IR[1] &
          IR[2] & IR[7], (IR[0] & ^IR[2]) + (IR[1] & ^IR[7]));
          COUT,ALUOUT = ALU(OP; ARGBUS; IDBUS; PS);
          P[6] <= ^(+/ALUOUT);
          P[0] <= (IDBUS[0] ! ALUOUT[0]) * (^IR[0] & ^IR[1] & ^IR[7],
          ^(^IR[0] & ^IR[1] & ^IR[7])); P[1] * (^IR[0] & ^IR[1] &
          ^IR[7]) <= IDBUS[1]; P[1] * ((IR[1] & IR[2] & IR[7]) &
          ((ARGBUS[0] & IDBUS[0] & ^ALUOUT[0]) + (^ARGBUS[0] &
          ^IDBUS[0] & ALUOUT[0]))) <= \1\; P[1] * ((IR[1] & IR[2] &
          IR[7]) & ^((ARGBUS[0] & IDBUS[0] & ^ALUOUT[0]) +
          (^ARGBUS[0] & ^IDBUS[0] & ALUOUT[0])))) <= \0\;
          P[7] * ((IR[0] + IR[2]) & IR[1]) <= COUT;
          => ((^IR[0] + IR[2]) & IR[7], IR[6], IR[0] & ^IR[1] &
          ^IR[6] & ^IR[7]) / (50, 51, 52).
57        DEADEND.
ENDSEQUENCE
CONTROLRESET(1);
          RESF * ^RES <= \1\;
          FF1 <= ^NMI; FF2 <= NMI + FF1;
          NMIFF * (^FF2 & ^NMI) <= \1\; NM = ^FF2 & ^NMI;
          P[1] * (^SO & PHI1) <= \1\; ADBUS = IADBUS; OSC = PHI0;
          PHI1 = OSC; PHI2 = ^OSC; CLOCK = PHI2 & ^ONE;
          ONE = ^RDY * (RW & PHI1); PS = P[4],P[7];
          ALUREG <= ALU[1:8](OP; ARGBUS; IDBUS; PS).
END.
```

APPENDIX G

COMBINATIONAL LOGIC UNIT DESCRIPTIONS

```
CLU:  ALUINT(OP; ARG1; ARG2; PS).
INPUTS:  OP[4]; ARG1[8]; ARG2[8]; PS[2].
OUTPUTS:  ALU[9].
CLUINTS:  BADD[9]  <: ADDER{8; 9}.
CLUINTS:  DADD[9]  <: DECADDER.
CTERMS:  OUT[9]; SAVE[9]; NARG2[8]; CONE; CZERO.

BODY
          CONE = \1\; CZERO = \0\; NARG2 = ^ARG2;
          SAVE = \0,0,0,0,0,0,0,0,0\;
          FOR N = 0 TO 10 CONSTRUCT
             IF N = 0 THEN
                OUT = BADD(ARG1; ARG2; CZERO)
             ELSE IF N = 1 THEN
                OUT = (BADD(ARG1; ARG2; PS[1]) & ^PS[0]) +
                      (DADD(OP[1]; ARG1; ARG2; PS[1]) & PS[0])
             ELSE IF N = 2 THEN
                OUT = \0\,(ARG1 & ARG2)
             ELSE IF N = 3 THEN
                OUT = \0\,(ARG1 + ARG2)
             ELSE IF N = 4 THEN
                OUT = \0\,(ARG1 @ ARG2)
             ELSE IF N = 5 THEN
                OUT = (BADD(ARG1; NARG2; PS[1]) & ^PS[0]) +
                      (DADD(OP[1]; ARG1; NARG2; PS[1]) & PS[0])
             ELSE IF N = 6 THEN
                OUT = BADD(ARG1; NARG2; CONE)
             ELSE IF N = 7 THEN
                OUT = ARG2,\0\
             ELSE IF N = 8 THEN
                OUT = ARG2[7],\0\,ARG2[0:6]
             ELSE IF N = 9 THEN
                OUT = ARG2,PS[1]
             ELSE IF N = 10 THEN
                OUT = ARG2[7],PS[1],ARG2[0:6]
             FI FI FI FI FI FI FI FI FI FI FI;
             ALU = (OUT & (&/TERM(N; OP))) + SAVE;
             SAVE = ALU
          ROF.
END.
```

```
CLU:  ADDER(OP1; OP2; CIN) {I; R}.
INPUTS:  OP1[I]; OP2[I]; CIN.
OUTPUTS:  SUM[R].
CLUNITS:  BITADDER[2]  <: FULLADDER.
CTERMS:  CARRY[R].

BODY
          CARRY[I] = CIN;
          FOR M = I - 1 TO 0 CONSTRUCT
             CARRY[M],SUM[M+1] =
                BITADDER(CARRY[M+1]; OP1[M]; OP2[M]);
             IF M = 0 THEN
                SUM[M] = CARRY[M]
             FI
          ROF.
END.
```

```
CLU:  DECADDER(COM; OP1; OP2; CIN).
INPUTS:  COM; OP1[8]; OP2[8]; CIN.
OUTPUTS:  DECOUT[9].
CLUNITS:  HBADD[5]  <: ADDER{4; 5}.
CTERMS:  CONST[4]; LOW[4]; UP[4]; CD1; CD2; CD3; CZERO.

BODY
          CZERO = \0\;
          CONST = (\0,1,1,0\ & ^COM) + (\1,0,1,0\ & COM);
          CD1,LOW = HBADD(OP1[4:7]; OP2[4:7]; CIN);
          CD2,DECOUT[5:8] = ((\0\,LOW) & ((^LOW[0] + (^LOW[1] &
             ^LOW[2])) & ^CD1 & ^COM)) + ((\1\,HBADD[1:4](LOW; CONST;
             CZERO)) & (((LOW[0] & LOW[1]) + (LOW[0] & LOW[2]) + CD1) &
             ^COM)) + ((CD1,LOW) & (CD1 & COM)) +
             ((CD1,HBADD[1:4](LOW; CONST; CZERO)) & (^CD1 & COM));
          CD3,UP = HBADD(OP1[0:3]; OP2[0:3]; CD2);
          DECOUT[0:4] = ((\0\,UP) & ((^UP[0] + (^UP[1] & ^UP[2])) &
             ^CD3 & ^COM)) + ((\1\,HBADD[1:4](UP; CONST; CZERO)) &
             (((UP[0] & UP[1]) + (UP[0] & UP[2]) + CD3) & ^COM)) +
             ((CD3,UP) & (CD3 & COM)) + ((CD3,HBADD[1:4](UP; CONST;
             CZERO)) & (^CD3 & COM)).
END.
```

```
CLU:  FULLADDER(CC; B1; B2).
INPUTS:  CC; B1; B2.
OUTPUTS:  OO[2].

BODY
            OO[1] = CC @ B1 @ B2;
            OO[0] = (B1 & B2) + ((B1 @ B2) & CC).
END.


CLU:  INCR(X).
INPUTS:  X[16].
OUTPUTS:  Y[16].

BODY
            FOR J = 15 TO 0 CONSTRUCT
                IF J = 15 THEN
                    Y[J] = ^X[J]
                ELSE
                    Y[J] = X[J] @ (&/X[J+1:15])
                FI
            ROF.
END.


CLU:  DECR(X).
INPUTS:  X[16].
OUTPUTS:  Y[16].

BODY
            FOR J = 15 TO 0 CONSTRUCT
                IF J = 15 THEN
                    Y[J] = ^X[J]
                ELSE
                    Y[J] = X[J] @ (&/^X[J+1:15])
                FI
            ROF.
END.
```

APPENDIX H

EXAMPLE OF SIMULATION

```
          AHPLMODULE:  MPU6502.
               MEMORY:  AC[8]; DL[8]; IR[8]; PC[16]; INR[16]; ALUREG[8];
                        SP[8]; X[8]; Y[8]; TR[8]; P[8]; RESF; IRQF; NMIF; FF1;
                        FF2; C; N; NMIFF; EXT; INH.
               EXINPUTS:  RES; IRQ; NMI; RDY; SO; PHIO.
               OUTPUTS:  SYNC; RW; PHI1; PHI2; ADBUS[16]; COUT; CIN; ALUOUT[8];
                         ONE; CD1; CD2; CD3; LOW[4]; UP[4]; DECOUT[9]; CONST[4].
               BUSES:  IDBUS[8]; IADBUS[16]; ARGBUS[8].
               EXBUSES:  DBUS[8].
               CLUNITS:  ADD1[9](ARGBUS; IDBUS; CIN);
                         INC[16](IADBUS); DEC[16](IADBUS);
                         ADD2[5](ARGBUS[4:7]; IDBUS[4:7]; CIN);
                         ADD3[5](ARGBUS[0:3]; IDBUS[0:3]; CD2);
                         ADD4[5](LOW; CONST); ADD5[5](UP; CONST).
1                    IADBUS = PC; SYNC = \1\; RW = \1\; INH <= \1\;
                     INR <= INC(IADBUS); NMIF <= \0\; IRQF <= \0\;
                     => (^RES) / (1).
2                    IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
                     INR <= INC(IADBUS); PC <= INR;
                     EXT * (RESF + NMIF + IRQF) <= \1\;
                     INH * ((^IR[0] & ^IR[3] & IR[4] & ^IR[5] & IR[6] & ^IR[7])
                     + (((^IR[0] & ^IR[2] & ^IR[3]) + (^IR[0] & ^IR[1] &
                     ^IR[3]) + (IR[1] & IR[2] & ^IR[3] & P[4])) & IR[4] & ^IR[5]
                     & ^IR[6] & IR[7])) <= \0\;
                     => ((RESF + NMIF + IRQF) & RES, ^RES) / (25, 1).
3                    NODELAY
                     => (IR[6] & IR[7]) / (21).
4                    NODELAY
                     => (^IR[0] & ^IR[3] & ^IR[5] & ^IR[6] & ^IR[7]) / (25).
5                    NODELAY
                     => (IR[4] & ^IR[5] & ^IR[7]) / (10).
6                    NODELAY
                     => ((^IR[5] & ^IR[7]) + (^IR[3] & IR[4] & ^IR[5])) / (12).
7                    NODELAY
                     => (IR[4]) / (16).
8                    NODELAY
                     => (IR[5]) / (19).
9                    NODELAY
                     => (44).
10                   NODELAY
                     => ((((^IR[0] + IR[1]) & IR[3]) + (IR[0] & IR[1] & IR[2])) &
                     IR[6]) / (21).
11                   NODELAY
                     => (30).
12                   NODELAY
```

```
              => (IR[0] & ^IR[1] & ^IR[2] & ^IR[3]) / (21).
13            NODELAY
              => (IR[0] & ^IR[1] & IR[2] & ^IR[3]) / (31).
14            NODELAY
              => (IR[6]) / (21).
15            NODELAY
              => (31).
16            NODELAY
              => (IR[0] & ^IR[1] & ^IR[2] & IR[3] & ^IR[7]) / (21).
17            NODELAY
              => (((^IR[0] & ^IR[1] & ^IR[2]) + ((^IR[0] + IR[1]) &
              IR[3])) & ^IR[6] & ^IR[7]) / (21).
18            NODELAY
              => (35).
19            NODELAY
              => ((((^IR[0] + IR[1]) & IR[3]) + (^IR[0] & ^IR[2]) +
              (^IR[0] & IR[1])) & ^IR[6] & ^IR[7]) / (21).
20            NODELAY
              => (40).
21            IADBUS = PC; IR <= DBUS; SYNC = \1\; RW = \1\;
              INR <= (INC(IADBUS) ! IADBUS) * (^(NMIF + (^IRQ & ^P[5])),
              (NMIF + (^IRQ & ^P[5]))); IRQF * (^IRQ & ^P[5]) <= \1\;
              NMIF * NMIFF <= \1\; NMIFF <= \0\;
              => (^RES, RES) / (1, 2).
22            IADBUS = INR; IR <= DBUS; SYNC = \1\; RW = \1\;
              INR <= (INC(IADBUS) ! IADBUS) * (^(NMIF + (^IRQ & ^P[5])),
              (NMIF + (^IRQ & ^P[5]))); IRQF * (^IRQ & ^P[5]) <= \1\;
              NMIF * NMIFF <= \1\; NMIFF <= \0\;
              => (^RES, RES) / (1, 2).
23            IADBUS = DL,TR; IR <= DBUS; SYNC = \1\; RW = \1\;
              INR <= (INC(IADBUS) ! IADBUS) * (^(NMIF + (^IRQ & ^P[5])),
              (NMIF + (^IRQ & ^P[5]))); IRQF * (^IRQ & ^P[5]) <= \1\;
              NMIF * NMIFF <= \1\; NMIFF <= \0\;
              => (^RES, RES) / (1, 2).
24            IADBUS = (INR[0:7],ALUREG ! ALUREG,INR[8:15]) * (^(N @ C),
              N @ C); IR <= DBUS; SYNC = \1\; RW = \1\;
              INR <= (INC(IADBUS) ! IADBUS) * (^(NMIF + (^IRQ & ^P[5])),
              (NMIF + (^IRQ & ^P[5]))); IRQF * (^IRQ & ^P[5]) <= \1\;
              NMIF * NMIFF <= \1\; NMIFF <= \C\;
              => (^RES, RES) / (1, 2).
25            IADBUS = \0,0,0,0,0,0,0,1\,SP; SYNC = \0\;
              INR[0:7] <= IADBUS[0:7];
              INR[8:15] <= (IADBUS[8:15] ! DEC[8:15](IADBUS) !
              INC[8:15](IADBUS)) * (^IR[1] & IR[2] & ^IR[4] & ^EXT,
              EXT + ((^IR[1] + IR[4]) & ^IR[2]),
              ((IR[1] & ^IR[4]) + (IR[2] & IR[4])) & ^EXT);
              RW = (\0\ ! \1\) * (((^IR[1] + IR[4]) & ^IR[2] & ^EXT) +
              IRQF + NMIF, (IR[2] & ^EXT) + RESF + (IR[1] & ^IR[4] &
```

```
         ^EXT)); DL * RW <= DBUS; DBUS = IDBUS * ^RW;
         IDBUS = (PC[0:7] ! INR[0:7] ! AC ! DL ! P) * (IRQF + NMIF,
         ^IR[1] & ^IR[2] & ^IR[4] & ^EXT, IR[1] & ^IR[2] & IR[4] &
         ^EXT, ^IR[1] & IR[2] & ^IR[4] & ^EXT, ^IR[1] & ^IR[2] &
         IR[4] & ^EXT); PC * (^IR[1] & ^IR[4] & ^EXT) <= INR;
         TR * (^IR[1] & IR[2] & ^IR[4] & ^EXT) <= IDBUS;
         => (^IR[2] & IR[4] & ^EXT & RES, ^IR[2] & IR[4] & ^EXT & RES,
         ^RES) / (21, 49, 1).
26       IADBUS = INR; SYNC = \0\; INR[8:15] <= (IADBUS[8:15] !
         INC[8:15](IADBUS) ! DEC[8:15](IADBUS)) * (IR[2] & IR[4] &
         ^EXT, IR[1] & ^IR[4] & ^EXT, (^IR[1] & ^IR[4]) + EXT);
         RW = (\0\ ! \1\) * ((^IR[1] & ^IR[4] & ^EXT) + IRQF + NMIF,
         (((IR[1] & ^IR[4]) + (IR[2] & IR[4])) & ^EXT) + RESF);
         DL * RW <= DBUS; DBUS = IDBUS * ^RW;
         IDBUS = (PC[0:7] ! PC[8:15]) * (^IR[1] & IR[2] & ^IR[4] &
         ^EXT, IRQF + NMIF + (^IR[1] & ^IR[2] & ^EXT));
         P[3] * IRQF <= \0\; P[3] * (^IR[1] & ^IR[2] & ^EXT) <= \1\;
         => (IR[2] & IR[4] & ^EXT & RES, IR[2] & IR[4] & ^EXT & RES,
         ^RES) / (21, 51, 1).
27       IADBUS = INR; SYNC = \0\; PC * (IR[1] & IR[2] & ^EXT) <= INR;
         INR[8:15] <= (INC[8:15](IADBUS) ! DEC[8:15](IADBUS)) *
         (IR[1] & ^IR[2] & ^EXT, ^IR[1] + EXT);
         RW = (\0\ ! \1\) * ((^IR[1] & ^EXT) + IRQF + NMIF,(IR[1] &
         ^EXT) + RESF); DL * RW <= DBUS; DBUS = IDBUS * ^RW;
         IDBUS = (PC[8:15] ! DL ! P) * (^IR[1] & IR[2] & ^EXT,
         IR[1] & ^EXT, (^IR[1] & ^IR[2] & ^EXT) + IRQF + NMIF);
         P * (IR[1] & ^IR[2] & ^EXT) <= IDBUS;
         TR * (IR[1] & IR[2] & ^EXT) <= IDBUS;
         => (^RES) / (1).
28       IADBUS = (\1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0\ !
         \1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0\ !
         \1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0\ !
         PC ! INR ! DL,TR) * (NMIF, RESF, IRQF +
         (^IR[1] & ^IR[2] & ^EXT), ^IR[1] & IR[2] & ^EXT,
         IR[1] & ^IR[2] & ^EXT, IR[1] & IR[2] & ^EXT); SYNC = \0\;
         RW = \1\; DL <= DBUS; PC * ((^IR[1] + ^IR[2]) & ^EXT) <= INR;
         INR * ((^IR[1] & ^IR[2]) + (IR[1] & IR[2]) + EXT) <=
         INC(IADBUS); IDBUS = (PC[8:15] ! DL) * (IR[1] & IR[2] & ^EXT,
         IR[1] & ^IR[2] & ^EXT); SP * (IR[1] & IR[2] & ^EXT) <= IDBUS;
         TR * (IR[1] & ^IR[2] & ^EXT) <= IDBUS;
         => (IR[1] & IR[2] & ^EXT & RES, (IR[1] @ IR[2]) & ^EXT & RES,
         (IR[1] @ IR[2]) & ^EXT & RES, ^RES) / (22, 23, 50, 1).
29       IADBUS = INR; SYNC = \0\; RW = \1\; DL <= DBUS; IDBUS = DL;
         TR <= IDBUS; P[5] <= \1\;
         RESF <= \0\; IRQF <= \0\; NMIF <= \0\; EXT <= \0\;
         => (^IR[1] & ^IR[2] & ^EXT & RES, ^IR[1] & ^IR[2] & ^EXT &
         RES, EXT & RES, ^RES) / (23, 50, 23, 1).
30       NODELAY
```

```
        => (^IR[0] & ^IR[3], ^(^IR[0] & ^IR[3]), \1\) / (57, 52, 21).
31      NODELAY
        => (^IR[3], ^IR[3]) / (22, 59).
32      NODELAY
        => ((^IR[0] & ^IR[1] & ^IR[2] & P[0]) + (^IR[0] & ^IR[1] &
        IR[2] & ^P[0]) + (^IR[0] & IR[1] & ^IR[2] & P[1]) +
        (^IR[0] & IR[1] & IR[2] & ^P[1]) + (IR[0] & ^IR[1] &
        ^IR[2] & P[7]) + (IR[0] & ^IR[1] & IR[2] & ^P[7]) +
        (IR[0] & IR[1] & ^IR[2] & P[6]) + (IR[0] & IR[1] &
        IR[2] & ^P[6])) / (22).
33      IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
        IDBUS = DL; N <= DL[0]; ARGBUS = IADBUS[8:15];
        C <= ADD1[0](ARGBUS; IDBUS; CIN);
        COUT = ADD1[0](ARGBUS; IDBUS; CIN); CIN = \0\;
        => (^(DL[0] @ COUT) & RES, ^RES) / (24, 1).
34      IADBUS = INR[0:7],ALUREG; DL <= DBUS;
        INR[8:15] <= IADBUS[8:15]; SYNC = \0\; RW = \1\;
        IDBUS = INR[0:7];
        ARGBUS = (\0,0,0,0,0,0,0,1\ ! \1,1,1,1,1,1,1,1\) *
        (^N & C, N & ^C); CIN = \0\;
        => (RES, ^RES) / (24, 1).
35      IADBUS = INR; DL <= DBUS; SYNC = \0\; RW = \1\;
        INR <= INC(IADBUS); IDBUS = DL;
        TR <= IDBUS; CIN = \0\;
        ARGBUS = (X ! Y) * ((IR[3] & IR[5] & ^IR[6]) + ((^IR[0] +
        IR[1]) & IR[3] & IR[6]), ^IR[5] + (IR[0] & ^IR[1] & IR[3] &
        IR[6])); C <= ADD1[0](ARGBUS; IDBUS; CIN);
        => (^IR[0] & IR[1] & ^IR[2] & ^IR[3] & ^IR[6] & ^IR[7] &
        RES, ^RES) / (23, 1).
36      IADBUS = (DL,ALUREG ! DL,TR) * (IR[3], ^IR[3]);
        SYNC = \0\; PC <= INR;
        RW = (\0\ ! \1\) * (IR[0] & ^IR[1] & ^IR[2] & ^IR[3],
        ^(IR[0] & ^IR[1] & ^IR[2] & ^IR[3])); DL * RW <= DBUS;
        DBUS = IDBUS * ^RW; IDBUS = (AC ! X ! Y ! DL) * (^RW &
        ^IR[6] & IR[7], ^RW & IR[6] & ^IR[7], ^RW & ^IR[6] &
        ^IR[7], IR[3]);
        INR <= (IADBUS ! INC(IADBUS)) * ((^IR[0] + IR[1]) & ^IR[3] &
        IR[6], ^IR[0] & IR[1] & ^IR[6] & ^IR[7]);
        INR[8:15] * IR[3] <= IADBUS[8:15];
        ARGBUS = (\0,0,0,0,0,0,0,0\ ! \0,0,C,0,0,0,0,1\) *
        (IR[3] & ^C, IR[3] & C); CIN = \0\;
        INH * ((^IR[0] & ^IR[1] & ^IR[3]) + (^IR[0] & ^IR[3] &
        IR[6]) + (^IR[0] & ^IR[2] & ^IR[3] & IR[7]) +
        (IR[1] & IR[2] & ^IR[3] & IR[7] &
        P[4]) + (((^IR[0] & ^IR[1] & IR[3]) + (^IR[0] & ^IR[2] &
        IR[3]) + (IR[1] & IR[2] & IR[3] & P[4])) & IR[7] & ^C))
        <= \0\;
        => (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] & RES, ((IR[0] &
```

```
            ^IR[1] & IR[2] & ^IR[3]) + (^IR[0] & ^IR[1] & ^IR[6] &
            ^IR[7]) + (IR[0] & IR[1] & ^IR[6] & ^IR[7]) + ((^IR[0] +
            IR[1]) & ^IR[3] & IR[7])) & RES,
            ((^(IR[0] & ^IR[1] & ^IR[2]) & IR[3] & ^IR[6]) + (IR[0] &
            ^IR[1] & IR[3] & IR[6])) & ^C & RES, (^IR[0] + IR[1]) &
            ^IR[3] & IR[6] & RES, (^IR[0] + IR[1]) & ^IR[3] & IR[6] &
            RES, ^RES) / (21, 48, 48, 57, 37, 1).
37          IADBUS = (INR ! ALUREG,INR[8:15]) * (^IR[3], IR[3]);
            SYNC = \0\; RW = (\0\ ! \1\) * ((^IR[3] &
            IR[6]) + (IR[0] & ^IR[1] & ^IR[2]), ^((^IR[3] &
            IR[6]) + (IR[0] & ^IR[1] & ^IR[2])));
            DL * RW <= DBUS; DBUS = IDBUS * ^RW; INR <= IADBUS;
            IDBUS = (AC ! DL) * (IR[0] & ^IR[1] & ^IR[2],
            ^IR[3] & ^IR[6]);
            TR <= IDBUS;
            INH * ((^IR[0] & IR[3] & IR[6]) + (^IR[0] & ^IR[1] &
            IR[3]) + (((^IR[0] & ^IR[2] & IR[3]) + (IR[1] & IR[2] &
            IR[3] & P[4])) & IR[7])) <= \0\;
            => (IR[0] & ^IR[1] & ^IR[2] & RES, ((^(IR[0] &
            ^IR[1] & ^IR[2]) & IR[3] & ^IR[6]) + (IR[0] & ^IR[1] &
            IR[6])) & RES, ^IR[3] & ^IR[6] & RES, (^IR[0] +
            IR[1]) & IR[3] & IR[6] & RES, (^IR[0] + IR[1]) & IR[3] &
            IR[6] & RES, ^RES) / (21, 48, 23, 57, 38, 1).
38          IADBUS = INR; SYNC = \0\; RW = \0\; DBUS = IDBUS;
            => (^IR[3] & RES, ^RES) / (21, 1).
39          IADBUS = INR; SYNC = \0\; RW = \0\; DBUS = IDBUS;
            => (RES, ^RES) / (21, 1).
40          IADBUS = \0,0,0,0,0,0,0,0\,DL; SYNC = \0\; PC <= INR;
            RW = (\0\ ! \1\) * (IR[0] & ^IR[1] & ^IR[2] & ^IR[3],
            ^(IR[0] & ^IR[1] & ^IR[2] & ^IR[3]));
            DL * RW <= DBUS; DBUS = IDBUS * ^RW;
            IDBUS = (AC ! X ! Y) * (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] &
            IR[7], (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] & IR[6]) +
            (IR[3] & ^IR[6]) + ((^IR[0] + IR[1]) & IR[3] & IR[6]),
            (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] & ^IR[6] & ^IR[7]) +
            (IR[0] & ^IR[1] & IR[3] & IR[6]));
            ARGBUS = IADBUS[8:15]; CIN = \0\; INR <= IADBUS;
            INH * ((^IR[0] & ^IR[3] & IR[6]) + (^IR[0] & ^IR[1] &
            ^IR[3]) + (^IR[0] & ^IR[2] & ^IR[3]) + (IR[1] & IR[2] &
            ^IR[3] & IR[7] & P[4])) <= \0\;
            => (IR[0] & ^IR[1] & ^IR[2] & ^IR[3] & RES, (((^IR[0] +
            IR[1]) & ^IR[3] & ^IR[6]) + (IR[0] & ^IR[1] & IR[2] &
            ^IR[3])) & RES, (^IR[0] + IR[1]) & ^IR[3] & IR[6] & RES,
            (^IR[0] + IR[1]) & ^IR[3] & IR[6] & RES, ^RES) /
            (21, 48, 57, 41, 1).
41          IADBUS = (INR ! INR[0:7],ALUREG) * (^IR[3], IR[3]);
            SYNC = \0\; RW = (\0\ ! \1\) * (^IR[3] & (IR[0] & ^IR[1] &
            ^IR[2]), ((^IR[0] + IR[1]) & IR[3]) + (IR[2] &
```

```
        IR[3])); DL * RW <= DBUS; DBUS = IDBUS * ^RW;
        IDBUS = (AC ! X ! Y) * (IR[0] & ^IR[1] & ^IR[2] & IR[7],
        IR[0] & ^IR[1] & ^IR[2] & IR[6], ^IR[2] & ^IR[6] & ^IR[7]);
        INR <= IADBUS; INH * ((^IR[0] & IR[3] & IR[6]) + (^IR[0] &
        ^IR[1] & IR[3]) + (^IR[0] & ^IR[2] & IR[3]) + (IR[1] &
        IR[2] & IR[3] & IR[7] & P[4])) <= \0\;
        => (IR[0] & ^IR[1] & ^IR[2] & IR[3] & RES, ((IR[0] &
        ^IR[1] & IR[2] & IR[3]) + ((^IR[0] + IR[1]) & IR[3] &
        ^IR[6])) & RES, (^IR[0] + IR[1]) & IR[3] & IR[6] & RES,
        (^IR[0] + IR[1]) & IR[3] & IR[6] & RES, ^RES) /
        (21, 48, 57, 42, 1).
42      IADBUS = INR; SYNC = \0\; RW = \0\; DBUS = IDBUS;
        => (^IR[3] & RES, ^RES) / (21, 1).
43      IADBUS = INR; SYNC = \0\; RW = \0\; DBUS = IDBUS;
        => (RES, ^RES) / (21, 1).
44      IADBUS = \0,0,0,0,0,0,0,0\,DL; SYNC = \0\; RW = \1\;
        DL <= DBUS; PC <= INR; INR[0:7] <= IADBUS[0:7];
        INR[8:15] <= INC[8:15](IADBUS); IDBUS = X;
        ARGBUS = IADBUS[8:15]; CIN = \0\;
        => (^RES) / (1).
45      IADBUS = (INR ! INR[0:7],ALUREG) * (IR[3], ^IR[3]);
        SYNC = \0\; RW = \1\; DL <= DBUS;
        INR[8:15] <= INC[8:15](IADBUS);
        IDBUS = DL; ARGBUS = Y; CIN = \0\;
        C <= ADD1[0](ARGBUS; IDBUS; CIN);
        => (^RES) / (1).
46      IADBUS = (INR ! DL,ALUREG) * (^IR[3], IR[3]);
        SYNC = \0\; RW = \1\; DL <= DBUS;
        IDBUS = DL;
        ARGBUS = (\0,0,0,0,0,0,0,0\ ! \0,0,0,0,0,0,0,1\) *
        (^C & IR[3], C & IR[3]); CIN = \0\;
        INR[8:15] <= IADBUS[8:15]; TR <= IDBUS;
        INH * (((^IR[0] & ^IR[1]) + (^IR[0] & ^IR[2]) + (IR[1] &
        IR[2] & P[4])) & IR[3] & ^C) <= \0\;
        => (^(IR[0] & ^IR[1] & IR[2]) & IR[3] & ^C & RES, ^RES) /
        (48, 1).
47      IADBUS = (ALUREG,INR[8:15] ! DL,TR) * (IR[3], ^IR[3]);
        SYNC = \0\;
        RW = (\0\ ! \1\) * (IR[0] & ^IR[1] & ^IR[2], ^(IR[0] &
        ^IR[1] & ^IR[2])); DL * RW <= DBUS; DBUS = IDBUS * ^RW;
        IDBUS = AC * (IR[0] & ^IR[1] & ^IR[2]);
        INH * ((^IR[0] & ^IR[2]) + (^IR[0] & ^IR[1]) + (IR[1] &
        IR[2] & P[4])) <= \0\;
        => (IR[0] & ^IR[1] & ^IR[2] & RES, ^RES) / (21, 1).
48      NODELAY
        => (21, 59).
49      PC <= INR;
        => (^RES) / (1).
```

```
50          IDBUS = PC[8:15]; SP <= IDBUS;
            => (60).
51          PC <= INR; IDBUS = DL; AC * IR[1] <= IDBUS;
            P * ^IR[1] <= IDBUS; ARGBUS = \0,0,0,0,0,0,0,0\; CIN = \0\;
            ALUOUT = ADD1[1:8](ARGBUS; IDBUS; CIN);
            P[0] * IR[1] <= ALUOUT[0]; P[6] * IR[1] <= ^(+/ALUOUT);
            => (RES, ^RES) / (50, 1).
52          P[7] * (^IR[0] & ^IR[1]) <= IR[2];
            P[5] * (^IR[0] & IR[1]) <= IR[2];
            P[4] * (IR[0] & IR[1] & IR[3]) <= IR[2];
            P[1] * (IR[0] & ^IR[1] & IR[2] & IR[3] & ^IR[6]) <= \0\;
            IDBUS = (AC ! X ! Y ! SP) * (^IR[1] & IR[2] & ^IR[3],
            (^IR[2] & IR[6]) + (IR[1] & IR[2] & ^IR[3]),
            (^IR[2] & ^IR[3] & ^IR[6]) + (IR[0] & ^IR[1] & ^IR[2] &
            ^IR[6]), IR[2] & IR[3] & IR[6]); CIN = \0\;
            ARGBUS = (\1,1,1,1,1,1,1,1\ ! \0,0,0,0,0,0,0,1\ !
            \0,0,0,0,0,0,0,0\) * ((^IR[1] & ^IR[2] &
            ^IR[3] & ^IR[6]) + (IR[1] & IR[6]), IR[1] &
            ^IR[3] & ^IR[6], (^IR[1] & IR[6]) + (IR[0] & ^IR[1] &
            ^IR[2] & IR[3]) + (^IR[1] & IR[2] & ^IR[3]));
            ALUOUT = ADD1[1:8](ARGBUS; IDBUS; CIN);
            P[0] * (^IR[3] + (IR[2] & IR[6]) + (IR[0] & ^IR[1] &
            ^IR[2] & ^IR[6])) <= ALUOUT[0];
            P[6] * (^IR[3] + (IR[2] & IR[6]) + (IR[0] & ^IR[1] &
            ^IR[2] & ^IR[6])) <= ^(+/ALUOUT);
            => (((IR[0] & ^IR[1] & ^IR[2] & IR[3] & ^IR[6]) + (^IR[1] &
            ^IR[2] & ^IR[3] & IR[6])) & RES, ((IR[1] & IR[2] & ^IR[3]) +
            (IR[1] & IR[6]) + (IR[2] & IR[6])) & RES, ((^IR[2] & ^IR[3]
            & ^IR[6]) + (^IR[1] & ^IR[3] & ^IR[6])) & RES, ^IR[2] &
            IR[3] & IR[6] & RES, (((^IR[0] + IR[1]) & IR[3]) + (IR[2] &
            IR[3] & ^IR[6])) & RES, ^RES) / (53, 54, 55, 56, 60, 1).
53          IDBUS = ALUREG; AC <= IDBUS;
            => (60).
54          IDBUS = ALUREG; X <= IDBUS;
            => (60).
55          IDBUS = ALUREG; Y <= IDBUS;
            => (60).
56          IDBUS = ALUREG; SP <= IDBUS;
            => (60).
57          IDBUS = (AC ! DL) * (^IR[5], IR[5]); CIN = \0\;
            ARGBUS = (\0,0,0,0,0,0,0,1\ ! \1,1,1,1,1,1,1,1\) *
            (IR[0] & IR[2], IR[0] & ^IR[2]);
            COUT,ALUOUT = (ADD1(ARGBUS; IDBUS; CIN) ! IDBUS,\0\ !
            IDBUS[7],\0\,IDBUS[0:6] ! IDBUS,P[7] !
            IDBUS[7],P[7],IDBUS[0:6]) * (INH, ^IR[1] & ^IR[2],
            ^IR[0] & IR[1] & ^IR[2], ^IR[1] & IR[2], ^IR[0] & IR[1] &
            IR[2]); P[0] <= ALUOUT[0]; P[6] <= ^(+/ALUOUT);
            P[7] * ^INH <= COUT; INH <= \1\; ALUREG * ^INH <= ALUOUT;
```

```
                        => (^IR[5] & RES, ^RES) / (53, 1).
          58            IDBUS = ALUREG;
                        => (60).
          59            CIN = (\0\ ! \1\ ! P[7]) * (IR[0] & ^IR[1] & IR[2], (IR[0] &
                        IR[1] & ^IR[2]) + (IR[0] & IR[1] & ^IR[7]), IR[1] & IR[2] &
                        IR[7]); IDBUS = (DL ! ^DL) * (^(IR[0] & IR[1]), IR[0] &
                        IR[1]); ARGBUS = (AC ! X ! Y ! \0,0,0,0,0,0,0,0\) * (^IR[0] +
                        (IR[1] & IR[7]), IR[1] & IR[2] & ^IR[7], ^IR[2] & ^IR[7],
                        IR[0] & ^IR[1]);
                        CD1,LOW = ADD2(ARGBUS[4:7]; IDBUS[4:7]; CIN);
                        CONST = (\0,1,1,0\ ! \1,0,1,0\) * (^IR[0], IR[0]);
                        CD2,DECOUT[5:8] = (\0\,LOW ! \1\,ADD4[1:4](LOW; CONST) !
                        CD1,LOW ! CD1,ADD4[1:4](LOW; CONST)) * ((^LOW[0] + (^LOW[1]
                        & ^LOW[2])) & ^CD1 & ^IR[0], ((LOW[0] & LOW[1]) + (LOW[0] &
                        LOW[2]) + CD1) & ^IR[0], CD1 & IR[0], ^CD1 & IR[0]);
                        CD3,UP = ADD3(ARGBUS[0:3]; IDBUS[0:3]; CD2);
                        DECOUT[0:4] = (\0\,UP ! \1\,ADD5[1:4](UP; CONST) ! CD3,UP !
                        CD3,ADD5[1:4](UP; CONST)) * ((^UP[0] + (^UP[1] & ^UP[2])) &
                        ^CD3 & ^IR[0], ((UP[0] & UP[1]) + (UP[0] & UP[2]) + CD3) &
                        ^IR[0], CD3 & IR[0], ^CD3 & IR[0]);
                        COUT,ALUOUT = (ADD1(ARGBUS; IDBUS; CIN) ! DECOUT !
                        \0\,(ARGBUS & IDBUS) ! \0\,(ARGBUS + IDBUS) !
                        \0\,(ARGBUS @ IDBUS)) * (INH, IR[1] & IR[2] & ^INH,
                        ^IR[0] & ^IR[1] & IR[2], ^IR[0] & ^IR[1] & ^IR[2], ^IR[0] &
                        IR[1] & ^IR[2]); P[6] <= ^(+/ALUOUT);
                        P[0] <= (IDBUS[0] ! ALUOUT[0]) * (^IR[0] & ^IR[1] & ^IR[7],
                        ^(^IR[0] & ^IR[1] & ^IR[7])); P[1] * (^IR[0] & ^IR[1] &
                        ^IR[7]) <= IDBUS[1]; P[1] * ((IR[1] & IR[2] & IR[7]) &
                        ((ARGBUS[0] & IDBUS[0] & ^ALUOUT[0]) + (^ARGBUS[0] &
                        ^IDBUS[0] & ALUOUT[0]))) <= \1\; P[1] * ((IR[1] & IR[2] &
                        IR[7]) & ^((ARGBUS[0] & IDBUS[0] & ^ALUOUT[0]) +
                        (^ARGBUS[0] & ^IDBUS[0] & ALUOUT[0]))) <= \0\;
                        P[7] * ((IR[0] + IR[2]) & IR[1]) <= COUT;
                        INH <= \1\; ALUREG * ^INH <= ALUOUT;
                        => ((^IR[0] + IR[2]) & IR[7] & RES, IR[6] & RES, IR[0] &
                        ^IR[1] & ^IR[6] & ^IR[7] & RES, ^RES) / (53, 54, 55, 1).
          60            DEADEND.
          ENDSEQUENCE
          CONTROLRESET(1);
                        RESF * ^RES <= \1\;
                        FF1 <= ^NMI; FF2 <= NMI + FF1;
                        NMIFF * (^FF2 & ^NMI) <= \1\;
                        P[1] * (^SO & PHI1) <= \1\; ADBUS = IADBUS;
                        PHI1 = PHI0; PHI2 = PHI0;
                        ONE = ^RDY * (RW & PHI1);
                        ALUREG * INH <= ADD1[1:8](ARGBUS; IDBUS; CIN).
          END.
```

```
        AHPLMODULE:  RAM1.
             MEMORY:  M1<256>[8].
             INPUTS:  ADBUS[16]; RW; PHI2.
             OUTPUTS:  E1[6].
             CLUNITS:  DCD[256](ADBUS[8:15]); BUSFN[8](M1;DCD).
        1            => (1).
        ENDSEQUENCE
        CONTROLRESET(1);
                     E1[0] = \1\;
                     E1[1] = ^ADBUS[0];
                     E1[2] = ^ADBUS[6];
                     E1[3] = \1\;
                     E1[4] = ^ADBUS[7];
                     E1[5] = PHI2;
                     DBUS = BUSFN(M1;DCD(ADBUS[8:15])) * ((&/E1) & RW);
                     M1 * (DCD(ADBUS[8:15]) & ((&/E1) & ^RW)) <= DBUS.
        END.
AHPL CIRCUIT DESCRIPTION MODULE NUMBER   3:        DATE: 30-Ju      TIME: 22:45
                                               (HPSIM/2.0 U OF ARIZ.)

        AHPLMODULE:  RAM2.
             MEMORY:  M2<256>[8].
             INPUTS:  ADBUS[16]; RW; PHI2.
             OUTPUTS:  E2[6].
             CLUNITS:  DCD[256](ADBUS[8:15]); BUSFN[8](M2;DCD).
        1            => (1).
        ENDSEQUENCE
        CONTROLRESET(1);
                     E2[0] = \1\;
                     E2[1] = ^ADBUS[0];
                     E2[2] = ^ADBUS[6];
                     E2[3] = \1\;
                     E2[4] = ADBUS[7];
                     E2[5] = PHI2;
                     DBUS = BUSFN(M2;DCD(ADBUS[8:15])) * ((&/E2) & RW);
                     M2 * (DCD(ADBUS[8:15]) & ((&/E2) & ^RW)) <= DBUS.
        END.
AHPL CIRCUIT DESCRIPTION MODULE NUMBER   4:        DATE: 30-Ju      TIME: 22:45
                                               (HPSIM/2.0 U OF ARIZ.)

        AHPLMODULE:  RAM3.
             MEMORY:  M3<256>[8].
             INPUTS:  ADBUS[16]; RW; PHI2.
             OUTPUTS:  E3[6].
             CLUNITS:  DCD[256](ADBUS[8:15]); BUSFN[8](M3;DCD).
        1            => (1).
```

```
          ENDSEQUENCE
          CONTROLRESET(1);
                         E3[0] = \1\;
                         E3[1] = ^ADBUS[0];
                         E3[2] = ADBUS[6];
                         E3[3] = \1\;
                         E3[4] = ^ADBUS[7];
                         E3[5] = PHI2;
                         DBUS = BUSFN(M3;DCD(ADBUS[8:15])) * ((&/E3) & RW);
                         M3 * (DCD(ADBUS[8:15]) & ((&/E3) & ^RW)) <= DBUS.
          END.
AHPL CIRCUIT DESCRIPTION MODULE NUMBER   5:          DATE: 30-Ju      TIME: 22:46
                                                 (HPSIM/2.0 U OF ARIZ.)

          AHPLMODULE:  ROM.
               MEMORY:  M4<1024>[8].
               INPUTS:  ADBUS[16]; PHI2.
               OUTPUTS:  E4[4].
               CLUNITS:  DCD[1024](ADBUS[6:15]); BUSFN[8](M4;DCD).
          1            => (1).
          ENDSEQUENCE
          CONTROLRESET(1);
                         E4[0] = PHI2;
                         E4[1] = ADBUS[0];
                         E4[2] = ADBUS[4];
                         E4[3] = ADBUS[5];
                         DBUS = BUSFN(M4;DCD(ADBUS[6:15])) * (&/E4).
          END.
HPSIM COMMUNICATION SECTION FOR THE ABOVE MODULES   DATE: 30-Ju   TIME: 22:46


          OPTION 6.
          CLOCKLIMIT 380.
          EXLINES  RES = 0#5,1;
                   IRQ = 1;
                   NMI = 1;
                   RDY = 1;
                   SO = 1;
                   PHIO = 1.
          INITIALIZE  M4<1023> <= 'FD; M4<1022> <= '00; M4<1021> <= '02;
                      M4<1020> <= '00; M1<0> <= 'FF; M1<2> <= 'FF;
                      M3<0:34> <= 'D8 ! 'A9 ! '00 ! '85 ! '01 ! '85 ! '03 ! '85 !
                      '04 ! '46 ! '02 ! '90 ! '0D ! '18 ! 'A5 ! '00 ! '65 ! '03 !
                      '85 ! '03 ! 'A5 ! '01 ! '65 ! '04 ! '85 ! '04 ! '06 ! '00 !
                      '26 ! '01 ! 'A5 ! '02 ! 'D0 ! 'E7 ! '00.
          OUTPUTS  PHIO; PHI1; PHI2; RES; IRQ; NMI; RDY; SO; ADBUS; IADBUS;
                   IDBUS; ARGBUS; DBUS; SYNC; RW; AC; DL; IR; PC; INR;
                   ALUREG; ALUOUT; SP; X; Y; P; TR.
```

```
463          DUMP  M1<0:255>;
464                M2<0:255>;
465                M3<0:255>;
466                M4<0:1023>.

::::: EXECUTION WILL STOP AFTER  380 CLOCK PULSES :::::
AHPL FUNCTION LEVEL SIMULATOR OUTPUT IS LISTED BELOW:
```

CLOCK #	PHI0	PHI1	PHI2	RES	IRQ	NMI	RDY	SO	ADBUS	IADBUS	IDBUS	ARGBUS	DBUS	SYNC	RW	AC	DL	IR	PC	INR	ALUREG	ALUOUT	SP	X	Y	P	TR
1	1	1	1	0	1	1	1	1	0000	0000	00	00	FF	1	1	00	00	00	0000	0000	00	00	00	00	00	00	00
2	1	1	1	0	1	1	1	1	0000	0000	00	00	FF	1	1	00	00	00	0000	0001	00	00	00	00	00	00	00
3	1	1	1	0	1	1	1	1	0000	0000	00	00	FF	1	1	00	00	00	0000	0001	00	00	00	00	00	00	00
4	1	1	1	0	1	1	1	1	0000	0000	00	00	FF	1	1	00	00	00	0000	0001	00	00	00	00	00	00	00
5	1	1	1	0	1	1	1	1	0000	0000	00	00	FF	1	1	00	00	00	0000	0001	00	00	00	00	00	00	00
6	1	1	1	1	1	1	1	1	0000	0000	00	00	FF	1	1	00	00	00	0000	0001	00	00	00	00	00	00	00
7	1	1	1	1	1	1	1	1	0001	0001	00	00	00	0	1	00	00	00	0000	0001	00	00	00	00	00	00	00
8	1	1	1	1	1	1	1	1	0100	0100	00	00	00	0	1	00	00	00	0001	0002	00	00	00	00	00	00	00
9	1	1	1	1	1	1	1	1	01FF	01FF	00	00	00	0	1	00	00	00	0001	01FF	00	00	00	00	00	00	00
10	1	1	1	1	1	1	1	1	01FE	01FE	00	00	00	0	1	00	00	00	0001	01FE	00	00	00	00	00	00	00
11	1	1	1	1	1	1	1	1	FFFC	FFFC	00	00	00	0	1	00	00	00	0001	01FD	00	00	00	00	00	00	00

12	1	1	1	1	1	1	1	1	FFFD	FFFD	00	00	02	0	1	00	00	00	0001	FFFD	00	00	00	00	00	00	00
13	1	1	1	1	1	1	1	1	0200	0200	00	00	D8	1	1	00	02	00	0001	FFFD	00	00	00	00	00	04	00
14	1	1	1	1	1	1	1	1	0201	0201	00	00	A9	0	1	00	02	D8	0001	0201	00	00	00	00	00	04	00
15	1	1	1	1	1	1	1	1	0201	0201	00	00	A9	1	1	00	A9	D8	0201	0202	00	00	00	00	00	04	00
16	1	1	1	1	1	1	1	1	0202	0202	00	00	00	0	1	00	A9	A9	0201	0202	00	00	00	00	00	04	00
17	1	1	1	1	1	1	1	1	0203	0203	00	00	85	1	1	00	00	A9	0202	0203	00	00	00	00	00	04	00
18	1	1	1	1	1	1	1	1	0204	0204	00	00	01	0	1	00	00	85	0202	0204	00	00	00	00	00	06	00
19	1	1	1	1	1	1	1	1	0001	0001	00	01	00	0	0	00	01	85	0204	0205	00	00	00	00	00	06	00
20	1	1	1	1	1	1	1	1	0205	0205	00	00	85	1	1	00	01	85	0205	0001	01	00	00	00	00	06	00
21	1	1	1	1	1	1	1	1	0206	0206	00	00	03	0	1	00	01	85	0205	0206	00	00	00	00	00	06	00
22	1	1	1	1	1	1	1	1	0003	0003	00	03	00	0	0	00	03	85	0206	0207	00	00	00	00	00	06	00
23	1	1	1	1	1	1	1	1	0207	0207	00	00	85	1	1	00	03	85	0207	0003	03	00	00	00	00	06	00
24	1	1	1	1	1	1	1	1	0208	0208	00	00	04	0	1	00	03	85	0207	0208	00	00	00	00	00	06	00
25	1	1	1	1	1	1	1	1	0004	0004	00	04	00	0	0	00	04	85	0208	0209	00	00	00	00	00	06	00
26	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	00	04	85	0209	0004	04	00	00	00	00	06	00
27	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	00	04	46	0209	020A	00	00	00	00	00	06	00
28	1	1	1	1	1	1	1	1	0002	0002	00	02	FF	0	1	00	02	46	020A	020B	00	00	00	00	00	06	00
29	1	1	1	1	1	1	1	1	0002	0002	FF	00	FF	0	0	00	FF	46	020B	0002	02	7F	00	00	00	06	00
30	1	1	1	1	1	1	1	1	0002	0002	7F	00	7F	0	0	00	FF	46	020B	0002	7F	00	00	00	00	05	00
31	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	00	FF	46	020B	0002	00	00	00	00	00	05	00
32	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	00	FF	90	020B	020C	00	00	00	00	00	05	00
33	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	00	0D	90	020C	020D	00	00	00	00	00	05	00
34	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	00	0D	18	020C	020E	00	00	00	00	00	05	00
35	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	00	A5	18	020E	020F	00	00	00	00	00	05	00
36	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	00	A5	A5	020E	020F	00	00	00	00	00	04	00
37	1	1	1	1	1	1	1	1	0000	0000	00	00	FF	0	1	00	00	A5	020F	0210	00	00	00	00	00	04	00
38	1	1	1	1	1	1	1	1	0210	0210	FF	00	65	1	1	00	FF	A5	0210	0000	00	FF	00	00	00	04	00
39	1	1	1	1	1	1	1	1	0211	0211	FF	00	03	0	1	00	FF	65	0210	0211	FF	00	00	00	00	84	00
40	1	1	1	1	1	1	1	1	0003	0003	00	03	00	0	1	FF	03	65	0211	0212	00	00	00	00	00	84	00
41	1	1	1	1	1	1	1	1	0212	0212	C0	FF	85	1	1	FF	00	65	0212	0003	03	FF	00	00	00	84	00
42	1	1	1	1	1	1	1	1	0213	0213	FF	00	03	0	1	FF	00	85	0212	0213	FF	00	00	00	00	84	00
43	1	1	1	1	1	1	1	1	0003	0003	FF	03	FF	0	0	FF	03	85	0213	0214	00	00	00	00	00	84	00
44	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	FF	03	85	0214	0003	02	00	00	00	00	84	00
45	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	FF	03	A5	0214	0215	00	00	00	00	00	84	00
46	1	1	1	1	1	1	1	1	0001	0001	00	01	00	0	1	FF	01	A5	0215	0216	00	00	00	00	00	84	00
47	1	1	1	1	1	1	1	1	0216	0216	00	00	65	1	1	FF	00	A5	0216	0001	01	00	00	00	00	84	00
48	1	1	1	1	1	1	1	1	0217	0217	00	00	04	0	1	FF	00	65	0216	0217	00	00	00	00	00	06	00
49	1	1	1	1	1	1	1	1	0004	0004	00	04	00	0	1	00	04	65	0217	0218	00	00	00	00	00	06	00
50	1	1	1	1	1	1	1	1	0218	0218	00	00	85	1	1	00	00	65	0218	0004	04	00	00	00	00	06	00
51	1	1	1	1	1	1	1	1	0219	0219	00	00	04	0	1	00	00	85	0218	0219	00	00	00	00	00	06	00
52	1	1	1	1	1	1	1	1	0004	0004	00	04	00	0	0	00	04	85	0219	021A	00	00	00	00	00	06	00
53	1	1	1	1	1	1	1	1	021A	021A	00	00	06	1	1	00	04	85	021A	0004	04	00	00	00	00	06	00
54	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	00	04	06	021A	021B	00	00	00	00	00	06	00
55	1	1	1	1	1	1	1	1	0000	0000	00	00	FF	0	1	00	00	06	021B	021C	00	00	00	00	00	06	00
56	1	1	1	1	1	1	1	1	0000	0000	FF	00	FF	0	0	00	FF	06	021C	0000	00	FE	00	00	00	06	00
57	1	1	1	1	1	1	1	1	0000	0000	FE	00	FE	0	0	00	FF	06	021C	0000	FE	00	00	00	00	85	00
58	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	00	FF	06	021C	0000	00	00	00	00	00	85	00
59	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	00	FF	26	021C	021D	00	00	00	00	00	85	00

60	1	1	1	1	1	1	1	1	0001	0001	00	01	00	0	1	00	01	26	021D	021E	00	00	00	00	00	85	00
61	1	1	1	1	1	1	1	1	0001	0001	00	00	00	0	0	00	00	26	021E	0001	01	01	00	00	00	85	00
62	1	1	1	1	1	1	1	1	0001	0001	01	00	01	0	0	00	00	26	021E	0001	01	00	00	0C	00	04	00
63	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	00	00	26	021E	0001	00	00	00	00	00	04	00
64	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	00	00	A5	021E	021F	00	00	00	0C	C0	04	00
65	1	1	1	1	1	1	1	1	0002	0002	00	02	7F	0	1	00	02	A5	021F	0220	00	00	00	00	00	04	00
66	1	1	1	1	1	1	1	1	0220	0220	7F	00	D0	1	1	00	7F	A5	0220	0002	02	7F	00	00	00	04	00
67	1	1	1	1	1	1	1	1	0221	0221	7F	00	E7	0	1	00	7F	D0	0220	0221	7F	00	00	00	00	04	00
68	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	7F	E7	D0	0221	0222	00	00	00	0C	C0	04	00
69	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	7F	00	D0	0221	0222	09	00	00	00	00	04	00
70	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	7F	00	46	0221	020A	00	00	00	00	00	04	00
71	1	1	1	1	1	1	1	1	0002	0002	00	02	7F	0	1	7F	02	46	020A	020B	00	00	00	00	00	04	00
72	1	1	1	1	1	1	1	1	0002	0002	7F	00	7F	0	0	7F	7F	46	020B	0002	02	3F	00	00	00	04	00
73	1	1	1	1	1	1	1	1	0002	0002	3F	00	3F	0	0	7F	7F	46	020B	0002	3F	00	00	CC	C0	05	00
74	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	7F	7F	46	020B	0002	00	00	00	00	00	05	00
75	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	7F	7F	90	020B	020C	00	00	00	00	00	05	00
76	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	7F	0D	90	020C	020D	00	00	00	00	00	05	00
77	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	7F	0D	18	020C	020E	00	00	00	00	00	05	00
78	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	7F	A5	18	020E	020F	00	00	00	00	00	05	00
79	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	7F	A5	A5	C20E	020F	00	00	00	00	C0	04	00
80	1	1	1	1	1	1	1	1	0000	0000	00	00	FE	0	1	7F	00	A5	020F	0210	00	00	0C	CC	C0	04	00
81	1	1	1	1	1	1	1	1	0210	0210	FE	00	65	1	1	7F	FE	A5	021C	0000	00	FE	00	00	00	04	00
82	1	1	1	1	1	1	1	1	0211	0211	FE	00	03	0	1	7F	FE	65	0210	0211	FE	00	00	00	00	84	00
83	1	1	1	1	1	1	1	1	0003	0003	00	03	FF	0	1	FE	03	65	0211	0212	00	00	00	00	00	84	00
84	1	1	1	1	1	1	1	1	0212	0212	FF	FE	85	1	1	FE	FF	65	C212	0003	03	FD	00	CC	C0	84	00
85	1	1	1	1	1	1	1	1	0213	0213	FD	00	03	0	1	FE	FF	85	0212	0213	FD	00	00	00	00	85	00
86	1	1	1	1	1	1	1	1	0003	0003	FD	03	FD	0	0	FD	03	85	C213	0214	00	00	00	00	00	85	00
87	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	FD	03	85	0214	0003	00	00	0C	00	00	85	00
88	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	FD	03	A5	0214	0215	00	00	00	00	00	85	00
89	1	1	1	1	1	1	1	1	0001	0001	00	01	01	0	1	FD	01	A5	0215	0216	00	00	00	00	00	85	00
90	1	1	1	1	1	1	1	1	0216	0216	01	00	65	1	1	FD	01	A5	C216	0001	01	01	00	0C	C0	85	00
91	1	1	1	1	1	1	1	1	0217	0217	01	00	04	0	1	FD	01	65	0216	0217	01	00	00	00	00	05	00
92	1	1	1	1	1	1	1	1	0004	0004	00	04	00	0	1	01	04	65	0217	0218	00	00	00	00	00	05	00
93	1	1	1	1	1	1	1	1	0218	021B	00	01	85	1	1	01	00	65	0218	0004	04	02	00	00	00	05	00
94	1	1	1	1	1	1	1	1	0219	0219	02	00	04	0	1	01	00	85	0218	0219	02	00	00	00	00	04	00
95	1	1	1	1	1	1	1	1	0004	0004	02	04	02	0	0	02	04	85	0219	021A	00	00	0C	CC	C0	04	00
96	1	1	1	1	1	1	1	1	021A	021A	00	00	06	1	1	02	04	85	021A	0004	06	00	00	00	00	04	00
97	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	02	04	06	021A	021B	00	00	00	00	00	04	00
98	1	1	1	1	1	1	1	1	0000	0000	00	00	FE	0	1	02	00	06	021B	021C	00	00	00	00	00	04	00
99	1	1	1	1	1	1	1	1	0000	0000	FE	00	FE	0	0	02	FE	06	021C	0000	00	FC	00	00	00	04	00
100	1	1	1	1	1	1	1	1	0000	0000	FC	00	FC	0	0	02	FE	06	021C	0000	FC	00	00	00	00	85	00
101	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	02	FE	06	C21C	0000	00	00	00	00	00	85	00
102	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	02	FE	26	021C	021D	00	00	00	00	00	85	00
103	1	1	1	1	1	1	1	1	0001	0001	00	01	01	0	1	02	01	26	021D	021E	00	00	00	00	00	85	00
104	1	1	1	1	1	1	1	1	0001	0001	01	00	01	0	0	02	01	26	021E	0001	01	03	00	00	00	85	00
105	1	1	1	1	1	1	1	1	0001	0001	03	00	03	0	0	02	01	26	C21E	0001	03	00	00	00	C0	04	00
106	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	02	01	26	021E	0001	00	00	00	00	00	04	00
107	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	02	01	A5	021E	021F	00	00	00	00	00	04	00

108	1	1	1	1	1	1	1	1	0002	0002	00	02	3F	0	1	02	02	A5	021F	0220	00	00	00	00	00	04	00
109	1	1	1	1	1	1	1	1	0220	0220	3F	00	D0	1	1	02	3F	A5	0220	0002	02	3F	00	00	00	04	00
110	1	1	1	1	1	1	1	1	0221	0221	3F	00	E7	0	1	02	3F	D0	0220	0221	3F	00	00	00	00	04	00
111	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	3F	E7	D0	0221	0222	00	00	00	00	00	04	00
112	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	3F	00	D0	0221	0222	09	00	00	00	00	04	00
113	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	3F	00	46	0221	020A	00	00	00	00	00	04	00
114	1	1	1	1	1	1	1	1	0002	0002	00	02	3F	0	1	3F	02	46	020A	020B	00	00	00	00	00	04	00
115	1	1	1	1	1	1	1	1	0002	0002	3F	00	3F	0	0	3F	3F	46	020B	0002	02	1F	00	00	00	04	00
116	1	1	1	1	1	1	1	1	0002	0002	1F	00	1F	0	0	3F	3F	46	020B	0002	1F	00	00	00	00	05	00
117	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	3F	3F	46	020B	0002	00	00	00	00	00	05	00
118	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	3F	3F	90	020B	020C	00	00	00	00	00	05	00
119	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	3F	0D	90	020C	020D	00	00	00	00	00	05	00
120	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	3F	0D	18	020C	020E	00	00	00	00	00	05	00
121	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	3F	A5	18	020E	020F	00	00	00	00	00	05	00
122	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	3F	A5	A5	020E	020F	00	00	00	00	00	04	00
123	1	1	1	1	1	1	1	1	0000	0000	00	00	FC	0	1	3F	00	A5	020F	0210	00	00	00	00	00	04	00
124	1	1	1	1	1	1	1	1	0210	0210	FC	00	65	1	1	3F	FC	A5	0210	0000	00	FC	00	00	00	04	00
125	1	1	1	1	1	1	1	1	0211	0211	FC	00	03	0	1	3F	FC	65	0210	0211	FC	00	00	00	00	84	00
126	1	1	1	1	1	1	1	1	0003	0003	00	03	FD	0	1	FC	03	65	0211	0212	00	00	00	00	00	84	00
127	1	1	1	1	1	1	1	1	0212	0212	FD	FC	85	1	1	FC	FD	65	0212	0003	03	F9	00	00	00	84	00
128	1	1	1	1	1	1	1	1	0213	0213	F9	00	03	0	1	FC	FD	85	0212	0213	F9	00	00	00	00	85	00
129	1	1	1	1	1	1	1	1	0003	0003	F9	03	F9	0	0	F9	03	85	0213	0214	00	00	00	00	00	85	00
130	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	F9	03	85	0214	0003	FC	00	00	00	00	85	00
131	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	F9	03	A5	0214	0215	00	00	00	00	00	85	00
132	1	1	1	1	1	1	1	1	0001	0001	00	01	03	0	1	F9	01	A5	0215	0216	00	00	00	00	00	85	00
133	1	1	1	1	1	1	1	1	0216	0216	03	00	65	1	1	F9	03	A5	0216	0001	01	03	00	00	00	85	00
134	1	1	1	1	1	1	1	1	0217	0217	03	00	04	0	1	F9	03	65	0216	0217	03	00	00	00	00	05	00
135	1	1	1	1	1	1	1	1	0004	0004	00	04	02	0	1	03	04	65	0217	0218	00	00	00	00	00	05	00
136	1	1	1	1	1	1	1	1	0218	0218	02	03	85	1	1	03	02	65	0218	0004	04	06	00	00	00	05	00
137	1	1	1	1	1	1	1	1	0219	0219	06	00	04	0	1	03	02	85	0218	0219	06	00	00	00	00	04	00
138	1	1	1	1	1	1	1	1	0004	0004	06	04	06	0	0	06	04	85	0219	021A	00	00	00	00	00	04	00
139	1	1	1	1	1	1	1	1	021A	021A	00	00	06	1	1	06	04	85	021A	0004	0A	00	00	00	00	04	00
140	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	06	04	06	021A	021B	00	00	00	00	00	04	00
141	1	1	1	1	1	1	1	1	0000	0000	00	00	FC	0	1	06	00	06	021B	021C	00	00	00	00	00	04	00
142	1	1	1	1	1	1	1	1	0000	0000	FC	00	FC	0	0	06	FC	06	021C	0000	00	F8	00	00	00	04	00
143	1	1	1	1	1	1	1	1	0000	0000	F8	00	F8	0	0	06	FC	06	021C	0000	F8	00	00	00	00	85	00
144	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	06	FC	06	021C	0000	00	00	00	00	00	85	00
145	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	06	FC	26	021C	021D	00	00	00	00	00	85	00
146	1	1	1	1	1	1	1	1	0001	0001	00	01	03	0	1	06	01	26	021D	021E	00	00	00	00	00	85	00
147	1	1	1	1	1	1	1	1	0001	0001	03	00	03	0	0	06	03	26	021E	0001	01	07	00	00	00	85	00
148	1	1	1	1	1	1	1	1	0001	0001	07	00	07	0	0	06	03	26	021E	0001	07	00	00	00	00	04	00
149	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	06	03	26	021E	0001	00	00	00	00	00	04	00
150	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	06	03	A5	021E	021F	00	00	00	00	00	04	00
151	1	1	1	1	1	1	1	1	0002	0002	00	02	1F	0	1	06	02	A5	021F	0220	00	00	00	00	00	04	00
152	1	1	1	1	1	1	1	1	0220	0220	1F	00	D0	1	1	06	1F	A5	0220	0002	02	1F	00	00	00	04	00
153	1	1	1	1	1	1	1	1	0221	0221	1F	00	E7	0	1	06	1F	D0	0220	0221	1F	00	00	00	00	04	00
154	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	1F	E7	D0	0221	0222	00	00	00	00	00	04	00
155	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	1F	00	D0	0221	0222	09	00	00	00	00	04	00

156	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	1F	00	46	C221	020A	00	00	00	00	00	04	00
157	1	1	1	1	1	1	1	1	0002	0002	00	02	1F	0	1	1F	02	46	020A	020B	00	00	0C	CC	C0	04	00
158	1	1	1	1	1	1	1	1	0002	0002	1F	00	1F	0	0	1F	1F	46	020B	0002	02	0F	00	00	00	04	00
159	1	1	1	1	1	1	1	1	0002	0002	0F	00	0F	0	0	1F	1F	46	020B	0002	0F	00	00	C0	00	05	00
160	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	1F	1F	46	020B	0002	00	00	00	00	00	05	00
161	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	1F	1F	90	020B	020C	00	00	00	00	00	05	00
162	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	1F	0D	90	020C	020D	00	00	00	00	00	05	00
163	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	1F	0D	18	C20C	020E	00	00	00	0C	C0	05	00
164	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	1F	A5	18	020E	020F	00	00	00	00	00	05	00
165	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	1F	A5	A5	020E	020F	00	00	00	00	00	04	00
166	1	1	1	1	1	1	1	1	0000	0000	00	00	F8	0	1	1F	00	A5	020F	0210	00	00	00	00	00	04	00
167	1	1	1	1	1	1	1	1	0210	0210	F8	00	65	1	1	1F	F8	A5	C21C	0000	00	F8	00	00	CC	04	00
168	1	1	1	1	1	1	1	1	0211	0211	F8	00	03	0	1	1F	F8	65	0210	0211	F8	00	00	00	00	84	00
169	1	1	1	1	1	1	1	1	0003	0003	00	03	F9	0	1	F8	03	65	0211	0212	00	00	00	00	00	84	00
170	1	1	1	1	1	1	1	1	0212	0212	F9	F8	85	1	1	F8	F9	65	0212	0003	03	F1	00	00	00	84	00
171	1	1	1	1	1	1	1	1	0213	0213	F1	00	03	0	1	F8	F9	85	0212	0213	F1	00	00	00	00	85	00
172	1	1	1	1	1	1	1	1	0003	0003	F1	03	F1	0	0	F1	03	85	0213	0214	00	00	0C	CC	C0	85	00
173	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	F1	03	85	0214	0003	F4	00	00	00	00	85	00
174	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	F1	03	A5	0214	0215	00	00	00	00	C0	85	00
175	1	1	1	1	1	1	1	1	0001	0001	00	01	07	0	1	F1	01	A5	0215	0216	00	00	00	00	00	85	00
176	1	1	1	1	1	1	1	1	0216	0216	07	00	65	1	1	F1	07	A5	0216	0001	01	07	00	00	00	85	00
177	1	1	1	1	1	1	1	1	0217	0217	07	00	04	0	1	F1	07	65	0216	0217	07	00	00	00	00	05	00
178	1	1	1	1	1	1	1	1	0004	0004	00	04	06	0	1	07	04	65	0217	0218	00	00	00	00	C0	05	00
179	1	1	1	1	1	1	1	1	0218	0218	06	07	85	1	1	07	06	65	0218	0004	04	0E	00	00	00	05	00
180	1	1	1	1	1	1	1	1	0219	0219	0E	00	04	0	1	07	06	85	0218	0219	0E	00	00	00	00	04	00
181	1	1	1	1	1	1	1	1	0004	0004	0E	04	0E	0	0	0E	04	85	0219	021A	00	00	00	00	00	04	00
182	1	1	1	1	1	1	1	1	021A	021A	C0	00	06	1	1	0E	04	85	C21A	0004	12	00	00	00	00	04	00
183	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	0E	04	06	021A	021B	00	00	0C	CC	00	04	00
184	1	1	1	1	1	1	1	1	0000	0000	00	00	F8	0	1	0E	00	06	021B	021C	00	00	00	00	00	04	00
185	1	1	1	1	1	1	1	1	0000	0000	F8	00	F8	0	0	0E	F8	06	021C	0000	00	F0	00	00	00	04	00
186	1	1	1	1	1	1	1	1	0000	0000	F0	00	F0	0	0	0E	F8	06	021C	0000	F0	00	00	00	00	85	00
187	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	0E	F8	06	021C	0000	00	00	00	00	00	85	00
188	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	0E	F8	26	021C	021D	00	00	00	00	00	85	00
189	1	1	1	1	1	1	1	1	0001	0001	C0	01	07	0	1	0E	01	26	021D	021E	00	00	00	0C	C0	85	00
190	1	1	1	1	1	1	1	1	0001	0001	07	00	07	0	0	0E	07	26	021E	0001	01	0F	00	00	00	85	00
191	1	1	1	1	1	1	1	1	0001	0001	0F	00	0F	0	0	0E	07	26	021E	0001	0F	00	00	00	00	04	00
192	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	0E	07	26	021E	0001	00	00	00	00	00	04	00
193	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	0E	07	A5	C21E	021F	00	00	00	00	00	04	00
194	1	1	1	1	1	1	1	1	0002	0002	00	02	0F	0	1	0E	02	A5	021F	0220	00	00	0C	CC	C0	04	00
195	1	1	1	1	1	1	1	1	0220	0220	0F	00	D0	1	1	0E	0F	A5	0220	0002	02	0F	00	00	00	04	00
196	1	1	1	1	1	1	1	1	0221	0221	0F	00	E7	0	1	0E	0F	D0	0220	0221	0F	00	00	00	00	04	00
197	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	0F	E7	D0	0221	0222	00	00	00	00	00	04	00
198	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	0F	00	D0	0221	0222	09	00	00	00	00	04	00
199	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	0F	00	46	0221	020A	00	00	00	00	00	04	00
200	1	1	1	1	1	1	1	1	0002	0002	00	02	0F	0	1	0F	02	46	C20A	020B	00	00	00	CC	C0	04	00
201	1	1	1	1	1	1	1	1	0002	0002	0F	00	0F	0	0	0F	0F	46	020B	0002	02	07	00	00	00	04	00
202	1	1	1	1	1	1	1	1	0002	0002	07	00	07	0	0	0F	0F	46	020B	0002	07	00	00	00	00	05	00
203	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	0F	0F	46	020B	0002	00	00	00	00	00	05	00

204	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	0F	0F	90	020B	020C	00	00	00	00	00	05	00
205	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	0F	0D	90	020C	020D	00	00	00	00	00	05	00
206	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	0F	0D	18	020C	020E	00	00	00	00	00	05	00
207	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	0F	A5	18	020E	020F	00	00	00	00	00	05	00
208	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	0F	A5	A5	020E	020F	00	00	00	00	00	04	00
209	1	1	1	1	1	1	1	1	0000	0000	00	00	F0	0	1	0F	00	A5	020F	0210	00	00	00	00	00	04	00
210	1	1	1	1	1	1	1	1	0210	0210	F0	00	65	1	1	0F	F0	A5	0210	0000	00	F0	00	00	00	04	00
211	1	1	1	1	1	1	1	1	0211	0211	F0	00	03	0	1	0F	F0	65	0210	0211	F0	00	00	00	00	84	00
212	1	1	1	1	1	1	1	1	0003	0003	00	03	F1	0	1	F0	03	65	0211	0212	00	00	00	00	00	84	00
213	1	1	1	1	1	1	1	1	0212	0212	F1	F0	85	1	1	F0	F1	65	0212	0003	03	E1	00	CC	C0	84	00
214	1	1	1	1	1	1	1	1	0213	0213	E1	00	03	0	1	F0	F1	85	0212	0213	E1	00	00	00	00	85	00
215	1	1	1	1	1	1	1	1	0003	0003	E1	03	E1	0	0	E1	03	85	0213	0214	00	00	00	00	CC	85	00
216	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	E1	03	85	0214	0003	E4	00	00	00	00	85	00
217	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	E1	03	A5	0214	0215	00	00	00	00	00	85	00
218	1	1	1	1	1	1	1	1	0001	0001	00	01	0F	0	1	E1	01	A5	0215	0216	00	00	00	00	00	85	00
219	1	1	1	1	1	1	1	1	0216	0216	0F	00	65	1	1	E1	0F	A5	C216	0001	01	0F	00	00	C0	85	00
220	1	1	1	1	1	1	1	1	0217	0217	0F	00	04	0	1	E1	0F	65	0216	0217	0F	00	00	00	00	05	00
221	1	1	1	1	1	1	1	1	0004	0004	00	04	0E	0	1	0F	04	65	0217	0218	00	00	00	00	00	05	00
222	1	1	1	1	1	1	1	1	0218	0218	0E	0F	85	1	1	0F	0E	65	0218	0004	04	1E	00	00	00	05	00
223	1	1	1	1	1	1	1	1	0219	0219	1E	00	04	0	1	0F	0E	85	0218	0219	1E	00	00	00	00	04	00
224	1	1	1	1	1	1	1	1	0004	0004	1E	04	1E	0	0	1E	04	85	0219	021A	00	00	00	00	00	04	00
225	1	1	1	1	1	1	1	1	021A	021A	00	00	06	1	1	1E	04	85	021A	0004	22	00	00	00	00	04	00
226	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	1E	04	06	021A	021B	00	00	00	00	00	04	00
227	1	1	1	1	1	1	1	1	0000	0000	00	00	F0	0	1	1E	00	06	C21B	021C	00	00	00	00	00	04	00
228	1	1	1	1	1	1	1	1	0000	0000	F0	00	F0	0	0	1E	F0	06	021C	0000	00	E0	00	00	00	04	00
229	1	1	1	1	1	1	1	1	0000	0000	E0	00	E0	0	0	1E	F0	06	021C	0000	E0	00	00	00	00	85	00
230	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	1E	F0	06	021C	0000	00	00	00	00	00	85	00
231	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	1E	F0	26	C21C	021D	00	00	00	00	CC	85	00
232	1	1	1	1	1	1	1	1	0001	0001	00	01	0F	0	1	1E	01	26	021D	021E	00	00	00	00	00	85	00
233	1	1	1	1	1	1	1	1	0001	0001	0F	00	0F	0	0	1E	0F	26	021E	0001	01	1F	00	00	00	85	00
234	1	1	1	1	1	1	1	1	0001	0001	1F	00	1F	0	0	1E	0F	26	021E	0001	1F	00	00	00	00	04	00
235	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	1E	0F	26	C21E	0001	00	00	00	00	00	04	00
236	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	1E	0F	A5	021E	021F	00	00	00	00	00	04	00
237	1	1	1	1	1	1	1	1	0002	0002	00	02	07	0	1	1E	02	A5	021F	0220	00	00	00	00	00	04	00
238	1	1	1	1	1	1	1	1	0220	0220	07	00	D0	1	1	1E	07	A5	0220	0002	02	07	00	00	00	04	00
239	1	1	1	1	1	1	1	1	0221	0221	07	00	E7	0	1	1E	07	D0	022C	0221	07	00	00	00	00	04	00
240	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	07	E7	D0	0221	0222	00	00	0C	CC	00	04	00
241	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	07	00	D0	0221	0222	09	00	00	00	00	04	00
242	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	07	00	46	0221	020A	00	00	00	00	00	04	00
243	1	1	1	1	1	1	1	1	0002	0002	00	02	07	0	1	07	02	46	C20A	020B	00	00	00	00	00	04	00
244	1	1	1	1	1	1	1	1	0002	0002	07	00	07	0	0	07	07	46	020B	0002	02	03	0C	CC	00	04	00
245	1	1	1	1	1	1	1	1	0002	0002	03	00	03	0	0	07	07	46	020B	0002	03	00	00	00	00	05	00
246	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	07	07	46	020B	0002	00	00	00	00	00	05	00
247	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	07	07	90	020B	020C	00	00	00	00	00	05	00
248	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	07	0D	90	020C	020D	00	00	00	00	00	05	00
249	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	07	0D	18	020C	020E	00	00	00	00	00	05	00
250	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	07	A5	18	C20E	020F	00	00	00	00	C0	05	00
251	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	07	A5	A5	020E	020F	00	00	00	00	00	04	00

252	1	1	1	1	1	1	1	1	0000	0000	00	00	E0	0	1	07	00	A5	020F	0210	00	00	0C	CC	C0	04	00
253	1	1	1	1	1	1	1	1	0210	0210	E0	00	65	1	1	07	E0	A5	0210	0000	00	E0	00	00	00	04	00
254	1	1	1	1	1	1	1	1	0211	0211	E0	00	03	0	1	07	E0	65	0210	0211	E0	00	00	00	C0	84	00
255	1	1	1	1	1	1	1	1	0003	0003	00	03	E1	0	1	E0	03	65	0211	0212	00	00	00	00	00	84	00
256	1	1	1	1	1	1	1	1	0212	0212	E1	E0	85	1	1	E0	E1	65	0212	0003	03	C1	00	00	00	84	00
257	1	1	1	1	1	1	1	1	0213	0213	C1	00	03	0	1	E0	E1	85	0212	0213	C1	00	00	00	00	85	00
258	1	1	1	1	1	1	1	1	0003	0003	C1	03	C1	0	0	C1	03	85	0213	0214	00	00	00	0C	C0	85	00
259	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	C1	03	85	0214	0003	C4	00	00	C0	00	85	00
260	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	C1	03	A5	0214	0215	00	00	00	00	00	85	00
261	1	1	1	1	1	1	1	1	0001	0001	00	01	1F	0	1	C1	01	A5	0215	0216	00	00	00	00	00	85	00
262	1	1	1	1	1	1	1	1	0216	0216	1F	00	65	1	1	C1	1F	A5	0216	0001	01	1F	00	00	CC	85	00
263	1	1	1	1	1	1	1	1	0217	0217	1F	00	04	0	1	C1	1F	65	0216	0217	1F	00	00	00	00	05	00
264	1	1	1	1	1	1	1	1	0004	0004	00	04	1E	0	1	1F	04	65	0217	0218	00	00	00	00	00	05	00
265	1	1	1	1	1	1	1	1	0218	0218	1E	1F	85	1	1	1F	1E	65	0218	0004	04	3E	00	00	00	05	00
266	1	1	1	1	1	1	1	1	0219	0219	3E	00	04	0	1	1F	1E	85	C218	0219	3E	00	00	CC	CC	04	00
267	1	1	1	1	1	1	1	1	0004	0004	3E	04	3E	0	0	3E	04	85	0219	021A	00	00	0C	00	00	04	00
268	1	1	1	1	1	1	1	1	021A	021A	00	00	06	1	1	3E	04	85	021A	0004	42	00	00	00	00	04	00
269	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	3E	04	06	021A	021B	00	00	00	00	00	04	00
270	1	1	1	1	1	1	1	1	0000	0000	00	00	E0	0	1	3E	00	06	021B	021C	00	00	00	00	00	04	00
271	1	1	1	1	1	1	1	1	0000	0000	E0	00	E0	0	0	3E	E0	06	021C	0000	00	C0	0C	CC	00	04	00
272	1	1	1	1	1	1	1	1	0000	0000	C0	00	C0	0	0	3E	E0	06	021C	0000	C0	00	00	00	00	85	00
273	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	3E	E0	06	021C	0000	00	00	00	00	00	85	00
274	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	3E	E0	26	021C	021D	00	00	00	00	00	85	00
275	1	1	1	1	1	1	1	1	0001	0001	00	01	1F	0	1	3E	01	26	021D	021E	00	00	00	00	00	85	00
276	1	1	1	1	1	1	1	1	0001	0001	1F	00	1F	0	0	3E	1F	26	021E	0001	01	3F	00	00	00	85	00
277	1	1	1	1	1	1	1	1	0001	0001	3F	00	3F	0	0	3E	1F	26	021E	C001	3F	00	00	0C	C0	04	00
278	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	3E	1F	26	021E	0001	00	00	00	00	00	04	00
279	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	3E	1F	A5	021E	021F	00	00	00	00	00	04	00
280	1	1	1	1	1	1	1	1	0002	0002	00	02	03	0	1	3E	02	A5	021F	0220	00	00	00	00	00	04	00
281	1	1	1	1	1	1	1	1	0220	0220	03	00	D0	1	1	3E	03	A5	C220	0002	02	03	00	00	00	04	00
282	1	1	1	1	1	1	1	1	0221	0221	03	00	E7	0	1	3E	03	D0	0220	0221	03	00	0C	CC	CC	04	00
283	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	03	E7	D0	0221	0222	00	00	00	00	00	04	00
284	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	03	00	D0	0221	0222	09	00	00	00	CC	04	00
285	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	03	00	46	0221	020A	00	C0	00	00	00	04	00
286	1	1	1	1	1	1	1	1	0002	0002	00	02	03	0	1	03	02	46	C20A	020B	00	00	00	00	00	04	00
287	1	1	1	1	1	1	1	1	0002	0002	03	00	03	0	0	03	03	46	020B	0002	02	01	00	00	00	04	00
288	1	1	1	1	1	1	1	1	0002	0002	01	00	01	0	0	03	03	46	C20B	0002	01	00	00	00	00	05	00
289	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	03	03	46	020B	0002	00	00	CC	CC	00	05	00
290	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	03	03	90	020B	020C	00	00	00	00	00	05	00
291	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	03	0D	90	020C	020D	00	00	00	00	00	05	00
292	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	03	0D	18	020C	020E	00	00	0C	00	C0	05	00
293	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	1	1	03	A5	18	020E	020F	00	00	00	00	00	05	00
294	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	03	A5	A5	020E	020F	00	00	00	00	00	04	00
295	1	1	1	1	1	1	1	1	0000	0000	00	00	C0	0	1	03	00	A5	020F	C210	00	00	00	CC	C0	04	00
296	1	1	1	1	1	1	1	1	0210	0210	C0	00	65	1	1	03	C0	A5	0210	0000	00	C0	0C	CC	C0	04	00
297	1	1	1	1	1	1	1	1	0211	0211	C0	00	03	0	1	03	C0	65	0210	0211	C0	00	00	00	00	84	00
298	1	1	1	1	1	1	1	1	0003	0003	00	03	C1	0	1	C0	03	65	0211	0212	00	00	0C	00	00	84	00
299	1	1	1	1	1	1	1	1	0212	0212	C1	C0	85	1	1	C0	C1	65	0212	0003	03	81	00	00	00	84	00

300	1	1	1	1	1	1	1	1	0213	0213	81	00	03	0	1	C0	C1	85	0212	0213	81	C0	00	00	00	85	00
301	1	1	1	1	1	1	1	1	0003	0003	81	03	81	0	0	81	03	85	0213	0214	00	00	00	00	00	85	00
302	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	81	03	85	0214	0003	84	00	00	00	00	85	00
303	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	81	03	A5	0214	0215	00	00	00	00	00	85	00
304	1	1	1	1	1	1	1	1	0001	0001	00	01	3F	0	1	81	01	A5	0215	0216	00	00	00	00	00	85	00
305	1	1	1	1	1	1	1	1	0216	0216	3F	00	65	1	1	81	3F	A5	0216	0001	01	3F	00	00	00	85	00
306	1	1	1	1	1	1	1	1	0217	0217	3F	00	04	0	1	81	3F	65	0216	0217	3F	00	00	00	00	05	00
307	1	1	1	1	1	1	1	1	0004	0004	00	04	3E	0	1	3F	04	65	0217	0218	00	00	00	00	00	05	00
308	1	1	1	1	1	1	1	1	0218	0218	3E	3F	85	1	1	3F	3E	65	0218	0004	04	7E	00	00	00	05	00
309	1	1	1	1	1	1	1	1	0219	0219	7E	00	04	0	1	3F	3E	85	0218	0219	7E	00	00	00	00	04	00
310	1	1	1	1	1	1	1	1	0004	0004	7E	04	7E	0	0	7E	04	85	0219	021A	00	00	00	00	00	04	00
311	1	1	1	1	1	1	1	1	021A	021A	00	00	06	1	1	7E	04	85	021A	0004	82	00	00	00	00	04	00
312	1	1	1	1	1	1	1	1	021B	021B	00	00	00	0	1	7E	04	06	021A	021B	00	00	00	00	00	04	00
313	1	1	1	1	1	1	1	1	0000	0000	00	00	C0	0	1	7E	00	06	021B	021C	00	00	00	00	00	04	00
314	1	1	1	1	1	1	1	1	0000	0000	C0	00	C0	0	0	7E	C0	06	021C	0000	00	80	00	00	00	04	00
315	1	1	1	1	1	1	1	1	0000	0000	80	00	80	0	0	7E	C0	06	021C	0000	80	00	00	00	00	85	00
316	1	1	1	1	1	1	1	1	021C	021C	00	00	26	1	1	7E	C0	06	021C	0000	00	00	00	00	00	85	00
317	1	1	1	1	1	1	1	1	021D	021D	00	00	01	0	1	7E	C0	26	021C	021D	00	00	00	00	00	85	00
318	1	1	1	1	1	1	1	1	0001	0001	00	01	3F	0	1	7E	01	26	021D	021E	00	00	00	00	00	85	00
319	1	1	1	1	1	1	1	1	0001	0001	3F	00	3F	0	0	7E	3F	26	021E	0001	01	7F	00	00	00	85	00
320	1	1	1	1	1	1	1	1	0001	0001	7F	00	7F	0	0	7E	3F	26	021E	0001	7F	00	00	00	00	04	00
321	1	1	1	1	1	1	1	1	021E	021E	00	00	A5	1	1	7E	3F	26	021E	0001	00	00	00	00	00	04	00
322	1	1	1	1	1	1	1	1	021F	021F	00	00	02	0	1	7E	3F	A5	021E	021F	00	00	00	00	00	04	00
323	1	1	1	1	1	1	1	1	0002	0002	00	02	01	0	1	7E	02	A5	021F	0220	00	00	00	00	00	04	00
324	1	1	1	1	1	1	1	1	0220	0220	01	00	D0	1	1	7E	01	A5	0220	0002	02	01	00	00	00	04	00
325	1	1	1	1	1	1	1	1	0221	0221	01	00	E7	0	1	7E	01	D0	0220	0221	01	00	00	00	00	04	00
326	1	1	1	1	1	1	1	1	0222	0222	E7	22	00	0	1	01	E7	D0	0221	0222	00	00	00	00	00	04	00
327	1	1	1	1	1	1	1	1	0209	0209	00	00	46	1	1	01	00	D0	0221	0222	09	00	00	00	00	04	00
328	1	1	1	1	1	1	1	1	020A	020A	00	00	02	0	1	01	00	46	0221	020A	00	00	00	00	00	04	00
329	1	1	1	1	1	1	1	1	0002	0002	00	02	01	0	1	01	02	46	020A	020B	00	00	00	00	00	04	00
330	1	1	1	1	1	1	1	1	0002	0002	01	00	01	0	0	01	01	46	020B	0002	02	00	00	00	00	04	00
331	1	1	1	1	1	1	1	1	0002	0002	00	00	00	0	0	01	01	46	020B	0002	00	00	00	00	00	07	00
332	1	1	1	1	1	1	1	1	020B	020B	00	00	90	1	1	01	01	46	020B	0002	00	00	00	00	00	07	00
333	1	1	1	1	1	1	1	1	020C	020C	00	00	0D	0	1	01	01	90	020B	020C	00	00	00	00	00	07	00
334	1	1	1	1	1	1	1	1	020D	020D	00	00	18	1	1	01	0D	90	020C	020D	00	00	00	00	00	07	00
335	1	1	1	1	1	1	1	1	020E	020E	00	00	A5	0	1	01	0D	18	020C	020E	00	00	00	00	00	07	00
336	1	1	1	1	1	1	1	1	020E	020E	C0	00	A5	1	1	01	A5	18	020E	020F	00	00	00	00	00	07	00
337	1	1	1	1	1	1	1	1	020F	020F	00	00	00	0	1	01	A5	A5	020E	020F	00	00	00	00	00	06	00
338	1	1	1	1	1	1	1	1	0000	0000	00	00	80	0	1	01	00	A5	020F	0210	00	00	00	00	00	06	00
339	1	1	1	1	1	1	1	1	0210	0210	80	00	65	1	1	01	80	A5	0210	0000	00	80	00	00	00	06	00
340	1	1	1	1	1	1	1	1	0211	0211	80	00	03	0	1	01	80	65	0210	0211	80	00	00	00	00	84	00
341	1	1	1	1	1	1	1	1	0003	0003	00	03	81	0	1	80	03	65	0211	0212	00	00	00	00	00	84	00
342	1	1	1	1	1	1	1	1	0212	0212	81	80	85	1	1	80	81	65	0212	0003	03	01	00	00	00	84	00
343	1	1	1	1	1	1	1	1	0213	0213	01	00	03	0	1	80	81	85	0212	0213	01	00	00	00	00	45	00
344	1	1	1	1	1	1	1	1	0003	0003	01	03	01	0	0	01	03	85	0213	0214	00	00	00	00	00	45	00
345	1	1	1	1	1	1	1	1	0214	0214	00	00	A5	1	1	01	03	85	0214	0003	04	00	00	00	00	45	00
346	1	1	1	1	1	1	1	1	0215	0215	00	00	01	0	1	01	03	A5	0214	0215	00	00	00	00	00	45	00
347	1	1	1	1	1	1	1	1	0001	0001	00	01	7F	0	1	01	01	A5	0215	0216	00	00	00	00	00	45	00

```
348    1 1 1 1 1 1 1 1 0216 0216 7F 00 65 1 1 01 7F A5 0216 0001 01 7F 00 00 00 45 00
349    1 1 1 1 1 1 1 1 0217 0217 7F 00 04 0 1 01 7F 65 0216 0217 7F 00 00 00 00 45 00
350    1 1 1 1 1 1 1 1 0004 0004 00 04 7E 0 1 7F 04 65 0217 0218 00 00 00 00 00 45 00
351    1 1 1 1 1 1 1 1 0218 0218 7E 7F 85 1 1 7F 7E 65 0218 0004 04 FE 00 00 00 45 00
352    1 1 1 1 1 1 1 1 0219 0219 FE 00 04 0 1 7F 7E 85 0218 0219 FE 00 00 00 00 C4 00
353    1 1 1 1 1 1 1 1 0004 0004 FE 04 FE 0 0 FE 04 85 0219 021A 00 00 00 00 00 C4 00
354    1 1 1 1 1 1 1 1 021A 021A 00 00 06 1 1 FE 04 85 021A 0004 02 00 00 00 00 C4 00
355    1 1 1 1 1 1 1 1 021B 021B 00 00 00 0 1 FE 04 06 021A 021B 00 00 00 00 00 C4 00
356    1 1 1 1 1 1 1 1 0000 0000 00 00 80 0 1 FE 00 06 021B 021C 00 00 00 00 00 C4 00
357    1 1 1 1 1 1 1 1 0000 0000 80 00 80 0 0 FE 80 06 021C 0000 00 00 00 00 00 C4 00
358    1 1 1 1 1 1 1 1 0000 0000 00 00 00 0 0 FE 80 06 021C 0000 00 00 00 00 00 47 00
359    1 1 1 1 1 1 1 1 021C 021C 00 00 26 1 1 FE 80 06 021C 0000 00 00 00 00 00 47 00
360    1 1 1 1 1 1 1 1 021D 021D 00 00 01 0 1 FE 80 26 021C 021D 00 00 00 00 00 47 00
361    1 1 1 1 1 1 1 1 0001 0001 00 01 7F 0 1 FE 01 26 021D 021E 00 00 00 00 00 47 00
362    1 1 1 1 1 1 1 1 0001 0001 7F 00 7F 0 0 FE 7F 26 021E 0001 01 FF 00 00 00 47 00
363    1 1 1 1 1 1 1 1 0001 0001 FF 00 FF 0 0 FE 7F 26 021E 0001 FF 00 00 00 00 C4 00
364    1 1 1 1 1 1 1 1 021E 021E 00 00 A5 1 1 FE 7F 26 021E 0001 00 00 00 00 00 C4 00
365    1 1 1 1 1 1 1 1 021F 021F 00 00 02 0 1 FE 7F A5 021E 021F 00 00 00 00 00 C4 00
366    1 1 1 1 1 1 1 1 0002 0002 00 02 00 0 1 FE 02 A5 021F 0220 00 00 00 00 00 C4 00
367    1 1 1 1 1 1 1 1 0220 0220 00 00 D0 1 1 FE 00 A5 0220 0002 02 00 00 00 00 C4 00
368    1 1 1 1 1 1 1 1 0221 0221 00 00 E7 0 1 FE 00 D0 0220 0221 00 00 00 00 00 46 00
369    1 1 1 1 1 1 1 1 0222 0222 00 00 00 1 1 00 E7 D0 0221 0222 00 00 00 00 00 46 00
370    1 1 1 1 1 1 1 1 0223 0223 00 00 00 0 1 00 E7 00 0221 0223 00 00 00 00 00 46 00
371    1 1 1 1 1 1 1 1 0100 0100 02 00 02 0 0 00 00 00 0223 0224 00 00 00 00 00 46 00
372    1 1 1 1 1 1 1 1 01FF 01FF 24 00 24 0 0 00 00 00 0224 01FF 00 00 00 00 00 46 00
373    1 1 1 1 1 1 1 1 01FE 01FE 56 00 56 0 0 00 00 00 0224 01FE 00 00 00 00 00 56 00
374    1 1 1 1 1 1 1 1 FFFE FFFE 00 00 00 0 1 00 00 00 0224 01FD 00 00 00 00 00 56 00
375    1 1 1 1 1 1 1 1 FFFF FFFF 00 00 FD 0 1 00 00 00 01FD FFFF 00 00 00 00 00 56 00
376    1 1 1 1 1 1 1 1 FD00 FD00 FD 00 00 1 1 00 FD 00 01FD FFFF 00 00 00 00 00 56 00
377    1 1 1 1 1 1 1 1 FD01 FD01 00 00 00 0 1 00 FD 00 01FD FD01 00 00 FD 00 00 56 00
378    1 1 1 1 1 1 1 1 01FD 01FD FD 00 FD 0 0 00 00 00 FD01 FD02 00 00 FD 00 00 56 00
379    1 1 1 1 1 1 1 1 01FC 01FC 02 00 02 0 0 00 00 00 FD02 01FC 00 00 FD 00 00 56 00
380    1 1 1 1 1 1 1 1 01FB 01FB 56 00 56 0 0 00 00 00 FD02 01FB 00 00 FD 00 00 56 00

::::: PROGRAM REACHED THE CLOCKLIMIT. INTERPRETER STOPS :::::
HPSIM OUTPUT, DUMP OF MEMORIES AND REGISTERS:

M1          <   0:  35>  00 FF 00 01 FE 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1          <  36:  71>  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1          <  72: 107>  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1          < 108: 143>  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1          < 144: 179>  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1          < 180: 215>  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1        < 216: 251>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M1        < 252: 255>   00 00 00 00
M2        <   0:  35>   02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M2        <  36:  71>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M2        <  72: 107>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M2        < 108: 143>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M2        < 144: 179>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M2        < 180: 215>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M2        < 216: 251>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 56
M2        < 252: 255>   02 FD 56 24
M3        <   0:  35>   D8 A9 00 85 01 85 03 85 04 46 02 90 0D 18 A5 00 65 03
                        85 03 A5 01 65 04 85 04 06 00 26 01 A5 02 D0 E7 00 00
M3        <  36:  71>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M3        <  72: 107>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M3        < 108: 143>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M3        < 144: 179>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M3        < 180: 215>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M3        < 216: 251>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M3        < 252: 255>   00 00 00 00
M4        <   0:  35>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4        <  36:  71>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4        <  72: 107>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4        < 108: 143>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4        < 144: 179>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4        < 180: 215>   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 216: 251>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 252: 287>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 288: 323>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 324: 359>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 360: 395>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 396: 431>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 432: 467>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 468: 503>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 504: 539>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 540: 575>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 576: 611>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 612: 647>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 648: 683>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 684: 719>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 720: 755>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 756: 791>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 792: 827>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 828: 863>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 864: 899>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 900: 935>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 936: 971>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       < 972:1007>     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M4       <1008:1023>     00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 FD
```

LIST OF REFERENCES

1. Hill, F. J. and G. R. Peterson, Digital Systems: Hardware Organization and Design, John Wiley and Sons, Inc., 2nd Edition, 1978.

2. Patil, S. S. and T. A. Welch, "A Programmable Logic Approch for VLSI", IEEE Transaction on Computer, Vol. C-28, No. 9, pp. 594-601, September 1979.

3. MOS Technology, Inc., MCS6500 Microcomputer Family Hardware Manual, 2nd Edition, Norristown, PA., January 1976.

4. MOS Technology, Inc., MCS6500 Microcomputer Family Software Manual, 2nd Edition, Norristown, PA., January 1976.

5. De Jong, M. L., Programming & Interfacing the 6502 With Experiments, Howard W. Sams & Co.,Inc., Indianapolis, IN., 1980.

6. Navabi, Z. and F. J. Hill, "System Manual for AHPL Simulator(HPSIM2)", Design Language Research Memo #005, The Department of Electrical Engineering, The University of Arizona, June 1979.

7. Chen, D., "Compilation of Combinational Logic Units for Universal AHPL", M.S. Thesis, The University of Arizona, April 1981.

8. Hill, F. J. and G. R. Peterson, Introduction to Switching Theory & Logical Design, John Wiley and Sons, Inc., 3rd Edition, 1981.

9. Hill, F. J., "Clocked Storage Logic Arrays(SLA's)", Design Language Research Memo #007, The Department of Electrical Engineering, The University of Arizona, 1980.

10. Navabi, Z., "Stage 3 Row Ordering Algorithm", Design Language Research Memo, The Department of Electrical Engineering, The University of Arizona, June 1981.